

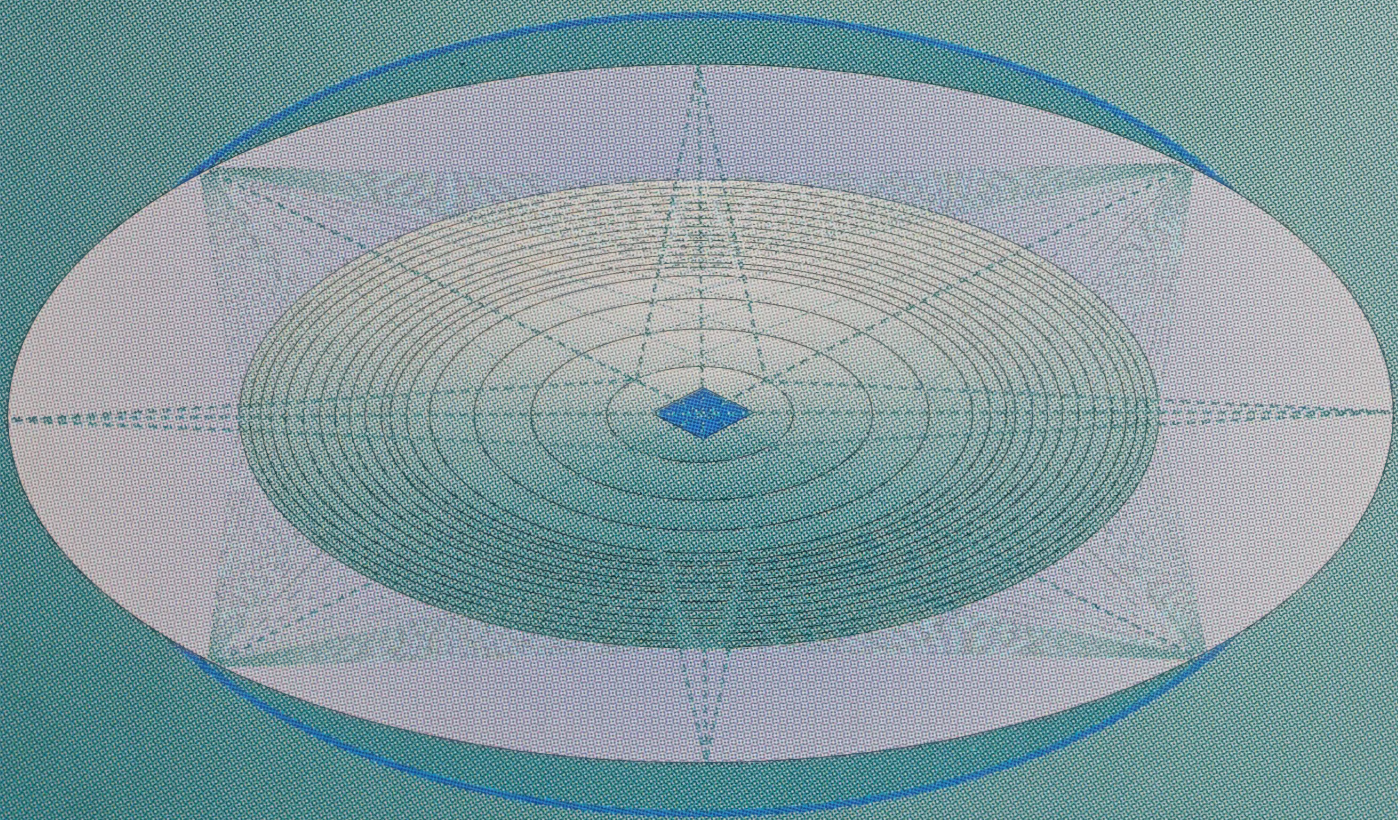


Carl Malamud

Includes  
TCP/IP and NFS



# ANALYZING **SUN** NETWORKS





# ANALYZING SUN NETWORKS

---

Carl Malamud

The Network File System (NFS) is the major alternative to the Open Software Foundation. With more than one million NFS computers already in existence, it is time for an authoritative volume that explains what NFS does and how it is used.

*Analyzing Sun Networks* presents a thorough overview of major industry standard protocols and architectures in the Network File System and TCP/IP environments. In addition to information for SUN workstation vendors and users the material applies to any vendor that has TCP/IP support—IBM, DEC, Sun, AT&T, Hewlett Packard, Novell, Cray, and many others.

This is the first book to describe NFS and the related protocols in the Open Network Computing (ONC) family. It covers the NFS, Remote Procedure Calls, License Servers, the Automounter, and a wide variety of other services in the ONC and Internet protocol families. There is also extensive discussion of TCP/IP and the underlying network on which TCP/IP is built. Numerous case studies of supercomputer sites, and Sun's own internal leading-edge network users give the discussion a real-world flavor. Throughout the book, extensive examples of actual network traffic are shown.


Also featured are chapters on many other topics that are critical in developing and implementing today's computer networks:

- Interoperability with OSI
- Messaging and X.400
- Naming services
- Security and public key encryption
- Network management and SNMP
- The Network Time Protocol

*(continued on back flap)*

**A VAN NOSTRAND REINHOLD BOOK**





Digitized by the Internet Archive  
in 2022 with funding from  
Public.Resource.Org

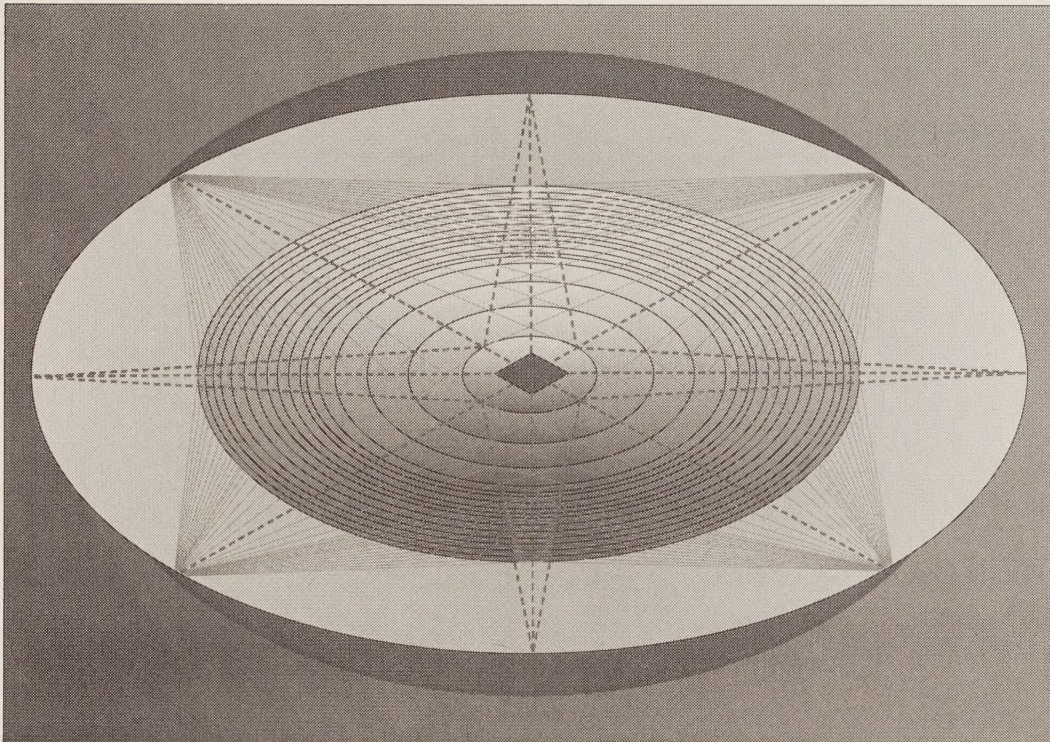
[https://archive.org/details/analyzingsunnetw00carl\\_0](https://archive.org/details/analyzingsunnetw00carl_0)







# Analyzing Sun Networks









# Analyzing Sun Networks

Carl Malamud



VAN NOSTRAND REINHOLD  
New York

Copyright © 1992 by Carl Malamud

Library of Congress Catalog Card Number 91-462

ISBN 0-442-00366-8

All rights reserved. No part of this work covered by the copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without written permission of the publisher.

Printed in the United States of America

Van Nostrand Reinhold  
115 Fifth Avenue  
New York, New York 10003

Chapman and Hall  
2-6 Boundary Row  
London, SE1 8HN, England

Thomas Nelson Australia  
102 Dodds Street  
South Melbourne 3205  
Victoria, Australia

Nelson Canada  
1120 Birchmount Road  
Scarborough, Ontario M1K 5G4, Canada

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Cover based on an original perspective drawing by Jan Vredeman de Vries (1527–1604), reprinted in *Perspective*, Dover Pictorial Archive Series (Dover Publications, 1968). Color was added without the participation of the original artist. Interior design by Carl Malamud.

### Library of Congress Cataloging-in-Publication Data

Malamud, Carl, 1959—

Analyzing Sun Networks / Carl Malamud.

564 p. cm.

Includes index.

ISBN 0-442-00375-7

1. Local area networks (Computer networks) 2. TCP/IP 3. Computer network architectures. 4. OSI (International standards) I. Title.

TK5105.7.M35 1990

621.39'81—dc20

91-462

CIP



# Contents

Preface	xi
A Note on Trademarks	xiii
Acknowledgments	xv
Chapter 1—Introduction	3
Overview	3
Chapter 2—Subnetworks	9
Introduction	9
Ethernet	12
IEEE LANs	15
Bridging Ethernets	21
FDDI and Token Ring	24
Token Ring Support	31
Wide-Area Links	31
SWAN: An Example	36
On the Internet	42
Packet Radio	53
For Further Reading	57
Chapter 3—Networks	61
Networks	61
The Internet Protocol	62
ISO CLNS and TCP's IP	68
Support Protocols: ARP and ICMP	69
ICMP	70

TCP	74
Connections and Options	84
TCP and OSI	88
UDP	90
RIP	90
OSPF and IS-IS	97
Exterior Gateway Protocols	100
Routing Between NSFnet and Milnet	105
TCP on a Sun	106
Starting a Connection	111
For Further Reading	112
 Chapter 4—STREAMS	 117
Protocol Stacks and the Operating System	117
Character I/O Mechanisms	118
The STREAMS Multiplexer	123
Transport Level Interface	123
For Further Reading	125
 Chapter 5—RPC and XDR	 129
Why an RPC?	129
The Portmapper	134
Asynchronous Mechanisms	140
The XDR Library Package	141
For Further Reading	148
 Chapter 6—Network File System	 151
NFS	151
The NFS Protocol	157
The NFS Implementation	163
NFS on the MVS Operating System	169
Hardware NFS Implementations	173
For Further Reading	177
 Chapter 7—RPC Tool	 181
RPC Compilers	181
Netwise RPC Tool	182



Procedure Declarations	185
Process Binding	187
The Client Stub	188
Server Control Procedure	189
Custom Procedures	192
ISO and Sun RPC Mechanisms	193
Other Interfaces to the Network	193
For Further Reading	196
 Chapter 8—Network Time Protocol	 199
Why Bother?	199
NTP	200
Associations	204
NTP Protocol	206
Handling Warped Servers	208
Implementation in the Internet	212
For Further Information	213
 Chapter 9—Names	 217
The New-Name Naming Service: <del>Yellow Pages</del> NIS	217
NIS	218
Typical NIS Operations	226
NIS+	226
Security in NIS+	235
Domain Name System	239
Servers	240
Resource Records	245
DNS Operation	249
Combining Name Services	256
For Further Reading	256
 Chapter 10—Security	 261
Secure Networks?	261
Basic Security	262
Security and RPC	263
Secure RPC	264
Kerberos	270

Administering Security	273
For Further Reading	277
 Chapter 11—The Automounter	 281
The Problem with /etc/fstab	281
Automounter Overview	282
Emulating an NFS Server	284
Maps	285
Multiple Mounts and Multiple Locations	287
Built-In Maps	291
When to Use the Automounter	292
For Further Reading	292
 Chapter 12—Distributed Applications	 295
Overview	295
License Service	296
License Service Operation	301
Network Lock Manager Protocol	303
REX	309
The R* Commands	313
The PC	315
For Further Reading	322
 Chapter 13—Mail	 325
Overview	325
SMTP	325
RFC 822	331
Using DNS to Forward Messages	338
UUCP	339
Munging Messages	341
Case Study: Sun's Mail System	342
X.400	347
IPMS	350
Message Transfer Agents	353
Distribution Lists	360
Message Stores	363
Gateways	364



Munging to X.400	365
For Further Reading	367
 Chapter 14—Files, Printers, and Terminals	 371
Overview	371
Telnet	371
The File Transfer Protocol	377
FTAM	386
Resource Location	393
Trivial FTP	396
Miscellaneous TCP/IP Services	398
The Line Printer Daemon	400
For Further Reading	404
 Chapter 15—Management	 407
Managing a Network	407
Structure of Management Information	409
Names	409
SNMP	412
SunNet Manager	415
The Management Database	418
The Discover Tool	421
SNMP Support	423
Operation of SunNet Manager	426
Basic Management Tools	429
For Further Reading	431
 Chapter 16—Interoperability	 435
Overview	435
The Special Case: SNA	436
SNA Case Studies	442
The Rest of the World	452
 Glossary	 461
 Index	 563







# Preface

Sun Microsystems has based their entire business on the concept of open systems. The original Sun workstation was built out of off-the-shelf parts. Using standard components to build state-of-the-art workstations and software is a policy that continues to this day.

This philosophy extends into networks. Sun networks are built on the industry standard TCP/IP protocols, the same protocols used for the international Internet network. This book shows how TCP/IP networks can be used to build heterogeneous, fast, and big networks.

TCP/IP is only the beginning for a network. Also explained are the Network File System, RPC, and all the other protocols that are part of the Open Networking Community (ONC) family. Because of the widespread acceptance of ONC—over one million computers on every major operating system—this book is about much more than just Sun networks; it is a book about any computers that use TCP/IP and NFS.

In addition to the industry standards, also included are network management, security, interoperability, and other topics that are vital to managing networks. We will see how OSI and SNA are built into the network and how this heterogeneous computing environment is managed.

Building viable network architectures is the theme of this book. Throughout the text, case studies from Sun's own network and other leading sites are used. One of the pleasures in writing this book has been working with technology that exists and is being used all over the world.

This is the last volume of a three-volume series on *Analyzing Networks*. For information on OSI-based networks, the reader is directed to *Analyzing DECnet/OSI Phase V*. For information on proprietary networks, the reader is directed to *Analyzing Novell Networks*. For information on heterogeneous networks that are operational today, turn the page.

Carl Malamud  
carl@malamud.com





# A Note on Trademarks

PostScript is a trademark of Adobe Systems.

Apple, AppleTalk, LocalTalk, and Macintosh are trademarks of Apple Computer.

dbase is a trademark of Ashton-Tate.

Ingres is a trademark of ASK, Inc.

STREAMS, Transport Layer Interface, Unix, and Unix System V Release 4 are trademarks of AT&T.

Auspex is a trademark of Auspex.

CRAY is the registered trademark of Cray Research, Inc.

Cullinet and IDMS are trademarks of Cullinet.

DEC, DECconnect, DECMcc, DECnet, DNA, Message Router, VAXcluster, and VMS are trademarks of Digital Equipment Corporation.

Epoch and InfiniteStorage are trademarks of Epoch.

AS/400, DISOSS, IBM, IBM PC LAN, PC/AT, PC/XT, PROFS, SNA, SNADS, System/370, System/38, and 3270 Display Station are trademarks of International Business Machines, Inc.

MS-DOS and Microsoft are trademarks of Microsoft.

Network General and Sniffer Analyzer are trademarks of Network General Corporation.

HYPERchannel is a trademark of Network Systems Corporation.

Network File System, NFS, Open Network Computing, ONC, SPARC, Sun, SunOS, and TOPS are trademarks of Sun Microsystems, Inc.

Sybase is a trademark of Sybase.

Carl Malamud is a trademark of Carl Malamud.



# Acknowledgments

Geoffrey Baehr, Director of Networking and Data Communications at Sun Microsystems, was instrumental in making the idea for a book on *Analyzing Sun Networks* become a reality. After previous encounters with large computer companies, I was delighted to find in Sun Microsystems a focus on technical excellence and open, nonproprietary systems. Special thanks goes to Terri Davis who provided assistance in ways too numerous to mention.

Reviews of the manuscript and answers to many, many questions were provided by the following engineers at Sun Microsystems: Sally Ahnger, Karl Auerbach, Gigi Babcock, Teresa Barr Beyer, Gabriele Cressman-Hirl, Linda Cwiak, Frank DeMarco, Jonathan Feiber, Dennis Freeman, Burt Fujii, Terence Gibson, Eric Johnson, Tom Kessler, Abhijit Khale, David Kipping, Fred Lowe, Karen Maleski, Milton Mallory, Mike Martinez, Dennis McLain, Chuck McManis, Bill Melohn, Bob Plummer, Ian Pope, Vipin Samar, Warren Smith, Warren Smith, Mark Stein, Richard Thio, Ian Vessey, Keith White, Yinpo Wong, and Dennis Yaro.

Bruce Nelson of Auspex, Joyce K. Reynolds of ISI, William T.C. Kramer and Louise D. Kokinakakis of the NASA Ames Research Center, Jay Dombrowski of the San Diego Supercomputer Center, Steve Crocker and Hilarie K. Orman of Trusted Information Systems, Robert Green of Unidata, and Dr. David Mills of the University of Delaware all provided valuable assistance.

Dr. Harry Saal of Network General once again let me use a Sniffer Network Analyzer to help bring protocols to life and provided a valuable review of the manuscript.

Brian Pawlowski of Sun Microsystems, provided an in-depth technical review of the entire manuscript. His attention to detail and his technical expertise contributed immensely to the final product. He asked me to add that he is “a genuinely warm and funny guy.” I would concur.

Dianne Littwin of Van Nostrand Reinhold has been the editor for all three volumes in this series. I’d like to express my sincere thanks to her for all her hard work and incredible patience. She deserves a vacation.





# Introduction





# Introduction

## Overview

For some, the network is the equivalent to a freeway. It's there, you have to use it, so you learn one or two exits and shuttle back and forth to work every day. For more inquisitive types, the freeway is just the backbone with a wide variety of different exits, each providing different services.

For this inquisitive reader, this book is intended to provide a road map. We assume the reader has some interest in knowing how a network based on the Transmission Control Protocol and the Internet Protocol (TCP/IP) operates.

TCP/IP is much broader than just Sun workstations, so the title of this book is a bit of a misnomer. The first portion of the book applies to any computer which supports TCP/IP, which is any computer still operating that doesn't use vacuum tubes.

We will see how TCP/IP can be used to integrate a variety of networks together into a coherent whole. Each network may be composed of a variety of different links—FDDI, Ethernet, X.25, ISDN, for example. These different data links are examined in more detail in Chapter 2. (The reader may also consult the glossary for quick explanations of acronyms.) Chapter 3 then shows how TCP and IP (and several other protocols) are built on top of the subnetworks, integrating them into internetworks.

Built on top of TCP/IP are a variety of services that were originally developed by Sun Microsystems; they are grouped together under the official name of Open Network Computing (ONC) or the unofficial moniker of the Network File System (NFS), a protocol that allows a file system on a remote computer to appear as a local drive. NFS opens the door for a wide variety of distributed computing, including the diskless workstation.

ONC is much more than NFS, however. It includes a remote procedure call protocol on which several other services are built. The Network Information Service (NIS) is a name service; there is a lock manager and many

other support services. Other applications, such as SunNet Manager or 3270 color screen emulation also use this ONC infrastructure.

ONC is the subject of much of this book. Chapters 5 and 7 discuss the RPC and XDR protocols, Chapter 6 describes NFS. Other ONC applications are also discussed: the Automounter in Chapter 11; the License Server and lock manager in Chapter 12.

Describing ONC and the TCP/IP protocols still doesn't really qualify this as a book on "Sun" networks. Although Sun developed NFS, a liberal licensing policy has seen it migrate to every major operating system.

This book is thus really about building heterogeneous networks with a slant on Sun workstations. Most of the technology described, however, has applicability to a wide selection of operating systems and hardware platforms.

The emphasis is on understanding how this heterogeneous network, of which the Sun is one type of server or workstation, works. A wide variety of topics are examined, organized in a way that attempts to make some sense out of the wide range of options that are available. The choice of topics is, of course, arbitrary. Not every issue is covered, and we have not attempted to deal with any issues in a definitive fashion. Points to sources for further reading are provided for people who want to implement software, tune systems, or do any of the other hands-on tasks of managing a network.

A fairly detailed overview does help, however, even if you will have to go chasing after some manual when a problem occurs. If you understand what pieces are involved, you have a valuable sense of perspective on the details listed in a reference manual or protocol specification. Throughout the book, illustrations bring concepts to life: network configurations, network analyzer screen dumps, lists of system calls, or reference tables. One reader termed this information "raw data," which this author has chosen to interpret in the good sense of the term: concrete examples that illustrate theory.

This raw data is presented to the reader because it gives an understanding of the capabilities of a given protocol or module. Looking at dumps from a network analyzer is useful for debugging error situations, but it is also useful as a learning tool to see how a packet is constructed. Knowing what system calls are available, or what management information is kept by a module, is equally instructive. Of course, all this raw data is supplemented by a healthy dose of commentary. A book of all pictures would be nice, but the picture book of network analysis will have to wait for another time.

The Sun Wide-Area Network (SWAN) is used throughout this book as a case study. In addition to the case study material from Sun, material from the San Diego Supercomputer Center, NASA-Ames, and the Internet is in-

PID	TT	STAT	TIME	COMMAND
0	?	D	0:00	swapper
1	?	IW	0:00	/sbin/init -s
2	?	D	0:00	pagedaemon
55	?	S	1:02	portmap
58	?	IW	0:00	ypbind
60	?	IW	0:00	keyserv
68	?	S	18:51	in.routed
71	?	I	0:00	(biode)
72	?	I	0:00	(biode)
73	?	I	0:00	(biode)
74	?	I	0:00	(biode)
84	?	IW	0:01	syslogd
92	?	IW	0:02	/usr/lib/sendmail -bd -q1h
96	?	IW	0:00	rpc.statd
98	?	IW	0:00	rpc.lockd
105	?	S	0:00	automount
112	?	S	10:46	update
115	?	IW	0:00	cron
120	?	IW	0:16	inetd
123	?	IW	0:00	/usr/lib/lpd
253	?	IW	0:00	rpc.rquotad
383	co	S	0:04	-csh (csh)
519	co	R	0:00	ps -ax

### 1-1 Processes on a Typical Workstation

cluded. Supplementing the case studies are a series of screen dumps taken from a Network General Sniffer Analyzer. This device is a modified PC that can be attached to token ring, Ethernet, ARCnet, and other networks to filter and capture data.

This captured data can then be analyzed. The Sniffer Analyzer is used because the information that is captured is displayed in a format that, given the technical material, is remarkably close to clear English.

For some readers, this book will provide all they need, a conceptual overview of how networking works in an environment with TCP/IP, NFS, or other protocols. Most readers, however, will probably have some contact with a Sun workstation or a TCP/IP network, or they wouldn't be reading this book. For those readers, this book is a supplement to the raw data of manuals, functional specifications, hardware diagrams, and all the other data that accompanies the computer. One nice thing about Sun is that much of this data is no longer shipped as large walls of documentation but is available electronically.



This book provides the overview. If the reader is a user of the network, the overview might help understand what is happening when a remote file system is mounted with NFS or why the license server is not letting you proceed with your application. Others will be interested in writing applications for use on the network. It is important to understand what tools are available so that an appropriate choice can be made. Using existing services, such as using NIS to register the location of a database server, means that the programmer doesn't have to reinvent the wheel.

Users and software developers are two classes that may find this information useful, but there is also one more class: the people who have to buy, install, and manage networks. For this class of readers, some material may be rudimentary compared to their current level of expertise. An engineer who designs FDDI chips is not going to learn anything about FDDI (or Ethernet) from reading this book but might learn something about message systems.

This is a bootstrap attempt: significantly more than a marketing white paper or the trade press but certainly less than all the primary resource documents. Hoping that human nature holds, the author is counting on the fact that a 1-inch thick book will be a more attractive target than the dozen or so linear feet that make up a decent set of source documents.

In Figure 1-1, the reader will see the first of many screen dumps. This dump is from a Sun workstation and is a listing of currently active processes on the computer. Eleven different processes in that listing are discussed in this book. An interesting test for the reader (and consequently of the author) is to see if, upon completion of the text, he or she can identify which processes are involved in a real network and what the processes do.

## CHAPTER 2

# Subnetworks





# Subnetworks

## Introduction

We start, as one might expect, at the beginning—in this case with a wire connecting two computers together. This wire may have two or many computers on it, may be fast or slow, and may be cheap or (usually) not. An examples of this service, termed here a *subnetwork*, is to exchange data between any two nodes on the wire.

The term *wire* is, of course, an abstract one. Two modems and a phone line provide the basic service described just as well as Ethernet, X.25, FDDI, token ring, ISDN, or any of the other subnetwork technologies available.

The TCP/IP world, like most network architectures, starts where this service leaves off. The IP protocol, the bottom of the TCP/IP stack, is a user of Ethernet, but so are many other network architectures such as OSI, Novell's NetWare, DECnet, and, in a funny sort of way, IBM's System Network Architecture (SNA).

A word on terminology. The word *subnetwork* has many different connotations in different architectures, as do words like *routers*, *gateways*, and *bridges*. In this book, the word *subnetwork* means any system that allows a host to think it can transmit data directly to another host without using the services of an intermediate node. When a host transmits data to another subnetwork, it uses the services of an intermediate node, called a *router*. A bridge is a device that, inside of a subnetwork, extends its reach, as in the case of an Ethernet-to-Ethernet bridge that connects two LANs together over a T1 leased line. A more precise term is MAC-layer bridge, indicating that the bridging is occurring at the medium access control sublayer of the data link layer.

The word *subnetwork* is a bit imprecise. It certainly means an Ethernet or FDDI LAN. However, it may also indicate any collection of nodes that appear to be locally connected. There may in fact be a series of routers in the middle, forwarding packets at the IP layer. In this chapter, subnet-

works are look at in the sense of the data link layer technologies like Ethernet or FDDI. The next chapter will show that when making routing decisions, an IP-based host is able to treat many different subnetworks as if they were all one big subnetwork.

The IP protocol has been used over most of the available subnetwork technologies: we will look at subnetworks ranging from amateur radio to FDDI to gigabit LAN and WAN systems. This tour is not meant to be exhaustive or comprehensive but simply illustrates the range of capabilities that are available.

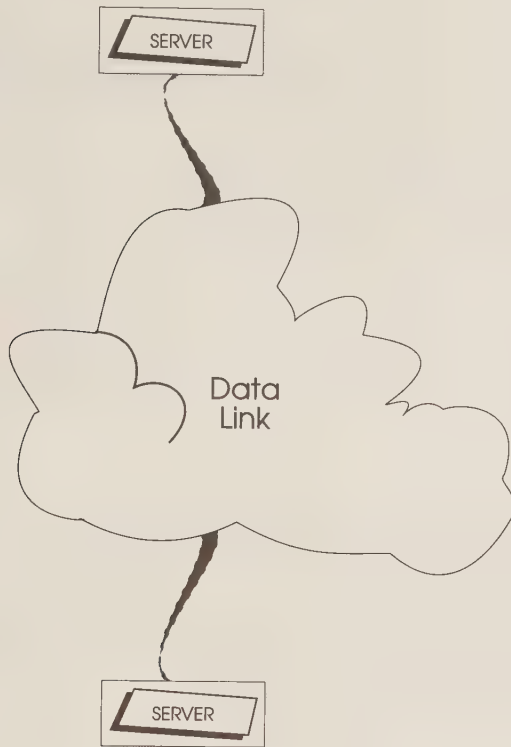
In addition to describing some basic subnetwork technologies, such as Ethernet or FDDI, this chapter has a series of case studies that show how combinations of subnetworks have been organized in several LAN and WAN environments. Again, these case studies are not meant to be exhaustive but are illustrative of how a few organizations decided to combine subnetworks. An important concept is that all the services described in the rest of this book work basically the same over any of these subnetworks.

Granted, fast networks perform operations faster, but the semantic content of an operation—writing a block of data using NFS, for example—remains the same. Errors may be different, as in the case of a connection aborting on a slow, noisy phone line, but if the operation succeeds, it works the same.

Matching a particular application to the lower layers that will support it can be difficult. Many organizations simply build an underlying infrastructure and then just monitor aggregate usage. Sometimes, particular applications are moved onto their own dedicated subnetworks. Typical NFS servers and clients share an Ethernet (proximity yielding better performance in this case), while two Cray computers may share a 1-Gbps Ultrane network to do their own crossmounts.

The IP concept of a subnetwork simply means that all traffic going in some direction can be directed to a single set of gateways. Those gateways will inspect the address more closely and choose how to actually route the packet. The basic service of all these technologies, however, is to move a packet of data between any two nodes connected to the subnetwork. In this sense, all subnetwork technologies can be viewed as a cloud (see Fig. 2-1) because the upper layers only rely on the basic service. The purpose of the data link and physical layers is to shield all the complexities of moving the packet from users of the service.

Throughout this book, we will see a variety of users of the data link service. The major user in a TCP/IP network is the Internet Protocol (IP) network layer. In addition to the network layer, however, there are other protocols, such as the Address Resolution Protocol or other network layers such as DECnet or OSI.



**2-1**  
A Cloud

Networks used to consist solely of data links meant to tie individual computers together. These point-to-point protocols were often meant to operate in a wide-area environment—this was in the days when computers were expensive and each site had one (“the”) computer. An example of these point-to-point protocols is IBM’s bisynchronous protocol (Bisynch). In the DEC environment, early PDP and VAX systems used the Digital Data Communications Message Protocol (DDCMP). In the TCP/IP world, protocols like Serial Line IP (SLIP) and the Point-to-Point Protocol (PPP) are used over dedicated connections.

In a local environment, point-to-point protocols were quickly supplanted by multiaccess LANs like the Ethernet protocols. Sun networks are Ethernet-centric: all Sun computers come with built-in Ethernet adapters and third-party systems such as terminal servers and print servers are all designed to fit on the Ethernet.

In addition to Ethernet, many systems are configured to use a faster LAN technology. HYPERchannel or Proteon are often used on systems running in the range of 50–100 Mbps, but many of these systems are being supplanted by FDDI networks. These high-speed data links are used as a backbone to connect different subnetworks together or for applications



2-2 A Bus

requiring high-speed data transfer as in the case of the graphic visualization of simulation results from a supercomputer.

In the wide-area environment, point-to-point connections are often used in conjunction with packet switched networks based on X.25 and other emerging technologies like frame relay, SMDS, and ISDN.

## Ethernet

Ethernet was originally developed at Xerox's Palo Alto Research Center and was subsequently standardized by the IEEE as the 802.3 LAN protocol. A note to the purist: this author uses the term *Ethernet* to refer to both the original Ethernet and the subsequent 802.3 standards. When it is necessary to distinguish between the two, 802.3 refers to the IEEE standard and Ethernet Level 2 refers to the latest version of the original DEC/Xerox/Intel specification.

The basic configuration of any Ethernet is a shared bus (see Fig. 2-2). The Ethernet looks to the user like a single wire with many nodes attached to it. Any one node can send a packet to any other node.

As we shall see shortly, the logical picture of a single bus or wire is an oversimplification of the actual physical configuration. Devices called repeaters are used to connect many different segments into a multisegment Ethernet. These multisegment Ethernets can in turn be connected using bridges into an extended Ethernet. The resulting combination of wires and connectors looks to the user like a logical bus; any node can send a packet to any other node.

## CSMA/CD Protocols

The basic operation of the Ethernet uses a protocol which goes under the catchy name of CSMA/CD, which stands for Carrier Sense-Multiple Ac-



Preamble		
Destination Address		48 bits
Source Address		
Protocol Type		Number not a valid length
Data		Data + Pad $\geq$ 46
Pad		
Frame Check Sequence		

**2-3**

An Ethernet Packet

cess/Collision Detect. Carrier Sense means that all nodes sharing the logical bus can sense if another node is transmitting. If one is, the well-behaved Ethernet node waits. Multiple Access means that if the medium is not in use, any node can send without waiting for special permission.

Since any two nodes can send, it is possible for two nodes to listen, sense that the medium is free, and simultaneously start sending the data. The result is a collision on the network, equivalent to two people talking at once. While two people may ignore the resulting collision, in an Ethernet both nodes detect the collision and stop sending. Both nodes then backoff for a random period of time (the period being different for the two nodes) and then check to see if the medium is free. If it is, the node can start sending.

While it is theoretically possible for nodes to keep on colliding, with nothing being sent, in reality the probability of repeated collisions is highly unlikely. A typical Ethernet has an average utilization of around 10 to 15 percent, sufficient for all nodes to send when they want to with very few collisions. It is not unusual, however, to have overloaded networks go over 50 percent average utilization—the penalty is in decreased performance.

If all of a network's nodes are on a single Ethernet, high growth of network utilization can indeed saturate the subnetwork. Realistically, however, most networks use multiple Ethernets. Nodes that frequently communicate with each other are put onto a single Ethernet. Then, either a bridge or a router is used to connect the multiple workgroups into a single integrated, extended LAN. The backbone of this LAN can be another Ethernet, or an FDDI subnetwork.

### *Ethernet Packet Formats*

Figure 2-3 shows basic format of an Ethernet Version 2 packet. Both the source and destination addresses have a 48-bit unique ID. These IDs are

universally administered and delegated to the manufacturers of Ethernet controller cards. This guarantees that every node will have a unique ID. For the reader who worries about running out of addresses, a 46-bit unique ID is equivalent to 70,254,592,000,000 unique addresses.

The first 2 bits of the ID are used as flags and are thus not used as part of the individual address. The first bit indicates if the address is an individual or group address. A group address is used to refer to many stations, as in the case of the address for all the NIS name servers on a single Ethernet subnetwork. An address of all 1s is a broadcast, which is received by every node on an Ethernet. The use of broadcasts is being increasingly discouraged because broadcasts are not generally useful for applications that span subnetworks and the indiscriminate use of broadcasts leads to network congestion.

The second bit of the ID indicates if the address is locally or universally administered. Universal addresses are guaranteed to be unique. It is possible, though not necessarily very smart, to locally administer the address space by having the network manager assign addresses. The reason this approach is not considered optimal is that it may lead to duplicate addresses (not to mention requiring somebody to come up with the unique addresses).

Following the source and destination addresses is the protocol type indicator. This indicator shows which user of the Ethernet service this particular packet is meant for. The protocol type indicator is how, for example, DECnet and TCP/IP can both share the services of a single Ethernet. The Ethernet module will deliver each packet to the appropriate user based on the protocol type field. Note the comment in Figure 2-3 which says the protocol type cannot be a valid length—this comment will become clearer when the 802.3 format is discussed.

The length of an Ethernet packet must be between 60 and 1524 bytes long. The minimum length requirement for an Ethernet packet ensures that it will remain on the medium long enough for nodes to detect a collision if one occurs.

The data part of the packet is transparent to the Ethernet module of the network. It could contain any data. Typically, it contains data for a user, such as the network layer. That data, in turn, will consist of a header for the network layer followed by some data (which will be a header for the transport layer and some data and so on).

The pad is used to pad out the data field so it reaches the minimum length requirement for the Ethernet. Following the pad is the frame check sequence (FCS), which is calculated based on the data being transmitted and appended to the end of the packet. The destination node will recalculate the FCS and then compare it to the one received. Ethernet is typically an error-free medium, so the FCS does not fail very often.

Preamble		
Destination Address		48 bits
Source Address		
Length		
Destination SAP		802.1 SNAP or "real" SAP
Source SAP		
PDU Type		XID
Protocol ID		Only if DSAP = SNAP
Data		
Pad		
Frame Check Sequence		

2-4

802.3 Packet Format

## IEEE LANs

Some aspects of the Ethernet frame format are protocol-dependent: other LAN technologies, such as token ring, do not need all the pieces of the Ethernet packet. For example, token ring has no need for a pad field.

When the IEEE began standardizing LAN technology, they broke Ethernet up into two sublayers:

- Logical Link Control (LLC)
- Medium Access Control (MAC)

The logical link control is medium independent and applies to all standardized LAN technologies, including token ring, token bus, CSMA/CD, and FDDI. Underneath the LLC sublayer and on top of the physical layer is the MAC sublayer. Each LAN technology has its own MAC format.

Figure 2-4 shows the packet format for an IEEE 802.3 LAN packet. Notice that the preamble, destination, and source addresses are the same as the Ethernet Level 2. This is the MAC portion of the field. After the address fields is a length field. Remember how the protocol type field in Ethernet was required to be a number that is not a valid length? This is how a node receiving a packet is able to determine which type of Ethernet packet it is and is thus able to interpret the fields that follow.

Following the length field is a destination service access point (SAP) indicator. The destination SAP is the beginning of the LLC portion of the packet and would be the same for all MAC technologies. This field is somewhat like the protocol type field in Ethernet Level 2.

There is a special SAP known as the SNAP, for subnetwork access point. The SNAP indicates that the address of the user is in the first byte of the



data field and the user data does not begin until the second byte. If the SAP contains a “real” address, the first byte of the data contains user data. The source service access point is the address of the program on the sending node which submitted the frame.

Finally, the LLC part of the header contains a protocol data unit (PDU) type field. There are two versions of the logical link control. In Class I, the type used by most networks, the operation is connectionless—each packet is a stand-alone unit of information.

The other mode of operation is the connection-oriented LLC Class II. In a connection-oriented mode, it is up to the data link layer to ensure that all packets in a stream of data are delivered to the destination SAP in the order they are sent. LLC Class II thus performs error detection and recovery.

LLC Class I is a best-effort delivery service. The packets are delivered, but there is no guarantee that a particular packet will make it or that two packets will be delivered in the order they are sent.

Within the LLC Class I operation, there are three types of packets. Frames sent by upper-layer service users are almost always Unnumbered Information (UI). The exchange ID (XID) and test frames are used for initialization and maintenance operations.

### *Ethernet Traffic*

Figures 2-5 and 2-6 show some typical Ethernet traffic as recorded on Network General’s Sniffer Analyzer. The Sniffer decodes each of the frames to show the headers of the different layers of a packet.

Figure 2-5 shows that the data link layer of the packet is a broadcast address. There is an individual source for the frame, followed by the protocol ID, which is the Address Resolution Protocol. Following the data link portion of the packet is the data field. The data is transparent to the Ethernet controller, which simply broadcasts it on the network. Whether the destination node exists or the destination user (the Ethertype) exists on the destination node are not questions addressed at this layer of the network.

Figure 2-6 shows how the Ethernet acts as a data highway, carrying traffic for different users. In fact, the view in this figure is a couple of layers above the Ethernet layer. The traffic going through is the Telnet virtual terminal service and the Transmission Control Protocol. Both of these services make use of the underlying Internet Protocol, which is the real user of the Ethernet.

### *Basic Ethernet Configuration*

To connect a device to the Ethernet, start with an Ethernet controller card. The original Ethernet medium was a piece of coaxial cable, also known as ThickWire or baseband. Access to the cable is with a device called a trans-



**DETAIL**

DLC: ----- DLC Header -----

DLC: Frame 2 arrived at 08:40:17.0749 ; frame size is 60 (003C hex) bytes.

DLC: Destination: BROADCAST FFFFFFFF, Broadcast

DLC: Source : Station Sun 0183E8

DLC: Ethertype = 0806 (ARP)

DLC:

ARP: ----- ARP/RARP frame -----

ARP: Hardware type = 1 (10Mb Ethernet)

ARP: Protocol type = 0800 (IP)

ARP: Length of hardware address = 6 bytes

ARP: Length of protocol address = 4 bytes

ARP: Opcode 1 (ARP request)

ARP: Sender's hardware address = Sun 0183E8

ARP: Sender's protocol address = [128.18.4.169]

ARP: Target hardware address = 000000000000

ARP: Target protocol address = [128.18.4.0]

ARP:

Frame 2 of 151

Use TAB to select windows

1 Help	2 Set mark	4 Zoom out	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	------------	---------	-------------------	--------------	--------------	----------------

## 2-5 ARP Traffic on an Ethernet

**SUMMARY**

	Delta T	DST	SRC	
54	0.0067	0000D0000070+Bridge00AC94	Telnet R PORT=25860	<0D><0A><0D>
55	0.0019	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252914
56	0.0082	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252919
57	0.0066	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252922
58	0.0148	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252922
59	0.0096	0000D0000070+Bridge00AC94	Telnet R PORT=25860	PTYPE=IBMPCA
60	0.0017	0000D0000070+Bridge00AC94	Telnet R PORT=25860	xon_xoff, sw
61	0.0118	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252932
62	0.0227	0000D0000070+Bridge00AC94	TCP D=25860 S=23	ACK=5258582

**DETAIL**

TCP: ----- TCP header -----

TCP: Source port = 25860

TCP: Destination port = 23 (Telnet)

TCP: Sequence number = 52585821

TCP: Acknowledgment number = 52529228

TCP: Data offset = 20 bytes

TCP: Flags = 10

TCP: ..0. .... = (No urgent pointer)

Frame 57 of 1354

Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

## 2-6 Multiple Users of the Ethernet

ceiver, and the transceiver is connected to the device (e.g., a workstation) with a transceiver cable.

Although baseband coaxial cable was the original medium, a variety of other media have been approved by the IEEE including:

- 10BASE5 (ThickWire) is standard coaxial cable. A single segment can be up to 500 meters long with 100 nodes.
- 10BASE2 (ThinWire) is a thinner coaxial cable—smaller maximum length but cheaper and easier to handle.
- 10BASET uses unshielded twisted pair.
- Fiber is used for security applications. Fiber is also used extensively for repeaters and bridges.
- CSMA/CD protocols can also run in a broadband environment, but sites are beginning to shift to fiber instead.

The details of configuring the specific media are beyond the scope of this book. The typical configuration, however, consists of several components:

- The medium
- A controller in the workstation or server
- A transceiver which is used to attach the medium
- A transceiver cable which is used to connect the transceiver to the controller

In many implementations, a multiport transceiver is used to connect several computers to the medium at one point of attachment. The multiport transceiver can be thought of as a concentrator.

The controller in the workstation varies by the type of peripheral bus (and hence which type of computer) as well as sustained throughput rate. Controllers also vary based on the type of physical wiring they support. Most controllers that support ThinWire also support the unshielded twisted pair (UTP) through the use of a device called a BALUN (balanced/unbalanced) which splices the ThinWire to the twisted pair.

In addition to standard computers, there are a wide variety of specialized devices which act as servers on the Ethernet. Print and terminal servers are two of the basic specialized servers that reside on the Ethernet. The terminal server is a cheap way of putting many terminals in contact with different hosts on the network. In addition to terminals, the terminal server can connect modems and printers.

Routers, covered in the next chapter, are one way of connecting several different Ethernets (or other subnetworks) together. The routers vary based on the number of connections they can support and the amount of throughput they have. As with other devices, routers are available from a wide variety of vendors.

### *Multisegment Ethernet*

A single segment of Ethernet is typically limited to 29 to 100 nodes. Twenty-nine is a typical maximum for ThinWire Ethernet segments (30 is the maximum, leaving one slot free for a repeater) and 100 is a typical maximum for a single segment of traditional coaxial cable. The Ethernet architecture supports up to 1024 nodes on a single subnetwork. The way this is done is by connecting multiple segments together using repeaters.

Figure 2-7 shows a multisegment Ethernet connected together using repeaters. The function of the repeater is to take all signals on one side of the repeater and retiming, reamplify, and retransmit it on the other segment. The repeater thus logically extends the wire at the physical layer of the network.

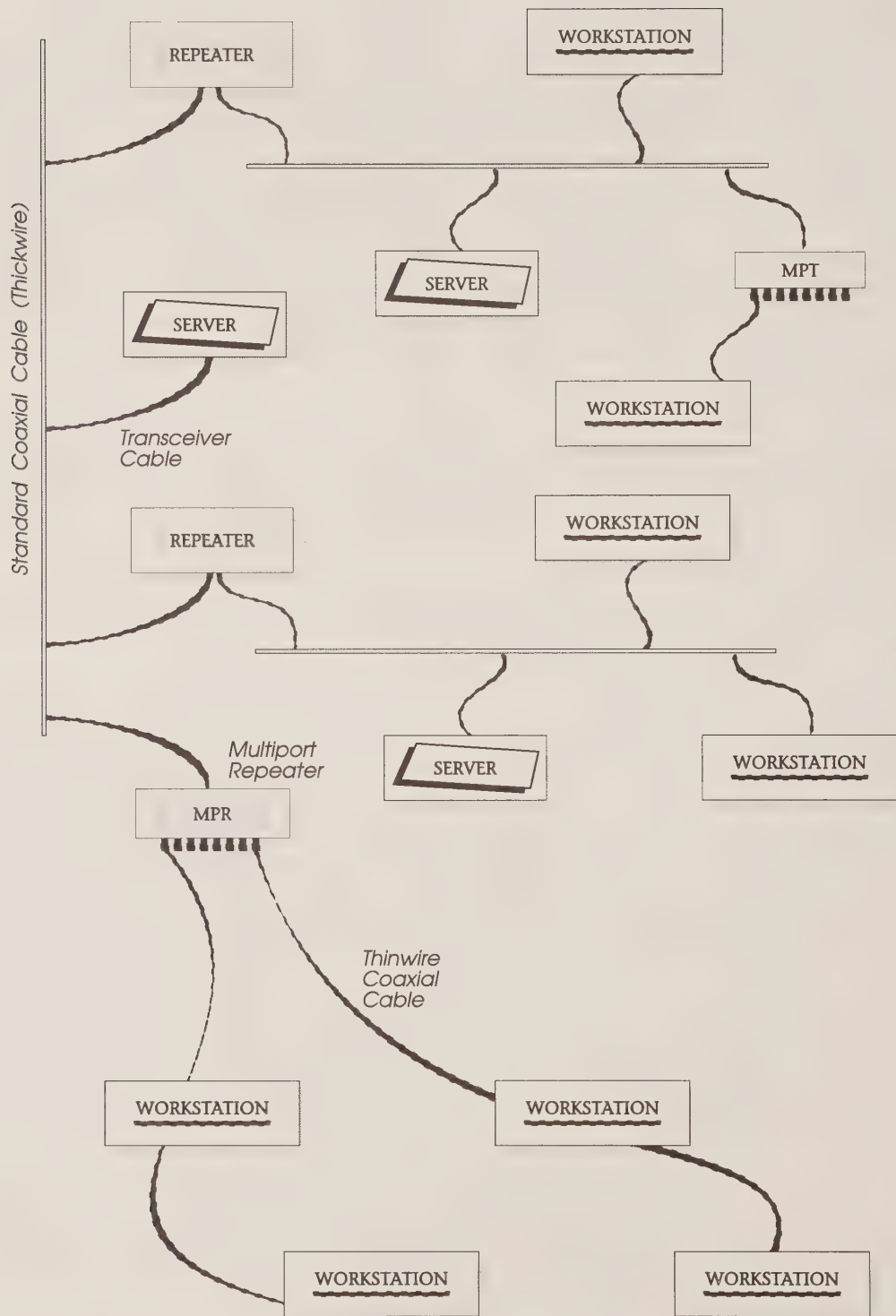
The basic rule of configuration is that there can be at most four repeaters in the path between any two nodes on the network. This means that a typical multisegment Ethernet consists of a backbone, with segments running from it. Nodes on two different segments would go through two repeaters when sending packets of data to each other.

The two-repeater rule is simply a means of ensuring that a collision will be detected within the architecturally defined parameters of the Ethernet protocol. Every device, including transceivers, repeaters, and the medium itself, introduces delay into the network.

Rules like the two-repeater rule are ways of making sure that there is not excessive delay and that the Ethernet will work properly. There are a variety of different rules, depending on the type of medium and the vendor promulgating the rule. In the case of DEC, the methodology for wiring the Ethernet has been codified in DECconnect. It is important to note that this is just one of many different available methodologies, all trying to do the same thing: provide a workable wiring for the Ethernet protocols. Any wiring system like DECconnect is really nothing more than a marketing umbrella for the collection of physical things that vendor happens to be selling at the time.

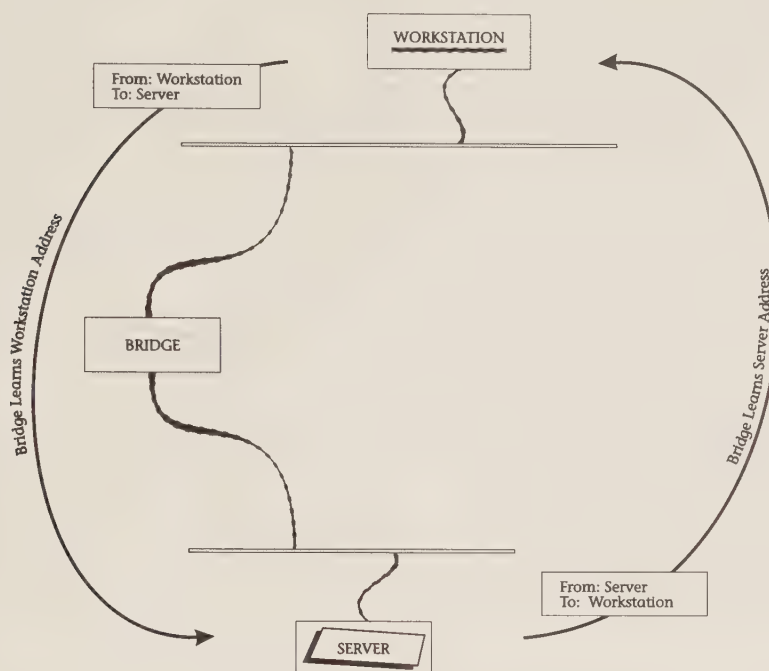
A variety of different repeaters are available. The typical repeater connects two ThickWire connections together using two transceiver cables. Multiport repeaters are typically used to connect one piece of ThickWire (the backbone) to up to eight segments of ThinWire or unshielded twisted pair.

A third type of repeater is the fiber repeater, which allows the segments being connected together to be 1000 meters or more apart. The fiber repeater is actually two half-repeaters, each connected to one of the segments. The fiber is then connected to each of the half-repeaters.



2-7 A Multisegment Ethernet



**2-8** Bridge Learning

## Bridging Ethernet

The bridge is a device that transparently connects multiple subnetworks (the subnetwork typically being a multisegment Ethernet) into an extended subnetwork. The user of the subnetwork sees the basic property of being able to transparently send one packet of data to any node on the subnetwork.

The difference between a bridge and a repeater is that the bridge only forwards those packets that need to be forwarded, while the repeater, operating at the physical level, retransmits every modulation of the signal. The bridge is thus a filter, selectively sending packets of data based on the source and destination addresses. In order to do the forwarding, the bridge must learn which nodes are on which side (see Fig. 2-8).

When a bridge initializes, it starts listening to each port, noting which source addresses it sees on each side. Based on this information, the bridge knows the location of some nodes. The bridge will not know about every node, however, because not all nodes will transmit information during the initialization period.

When the bridge sees a packet that has both destination and source addresses on the same side, it has no need to forward the packet, and ignores

it. If the bridge notices that a destination address is located on the other side, it forwards the packet.

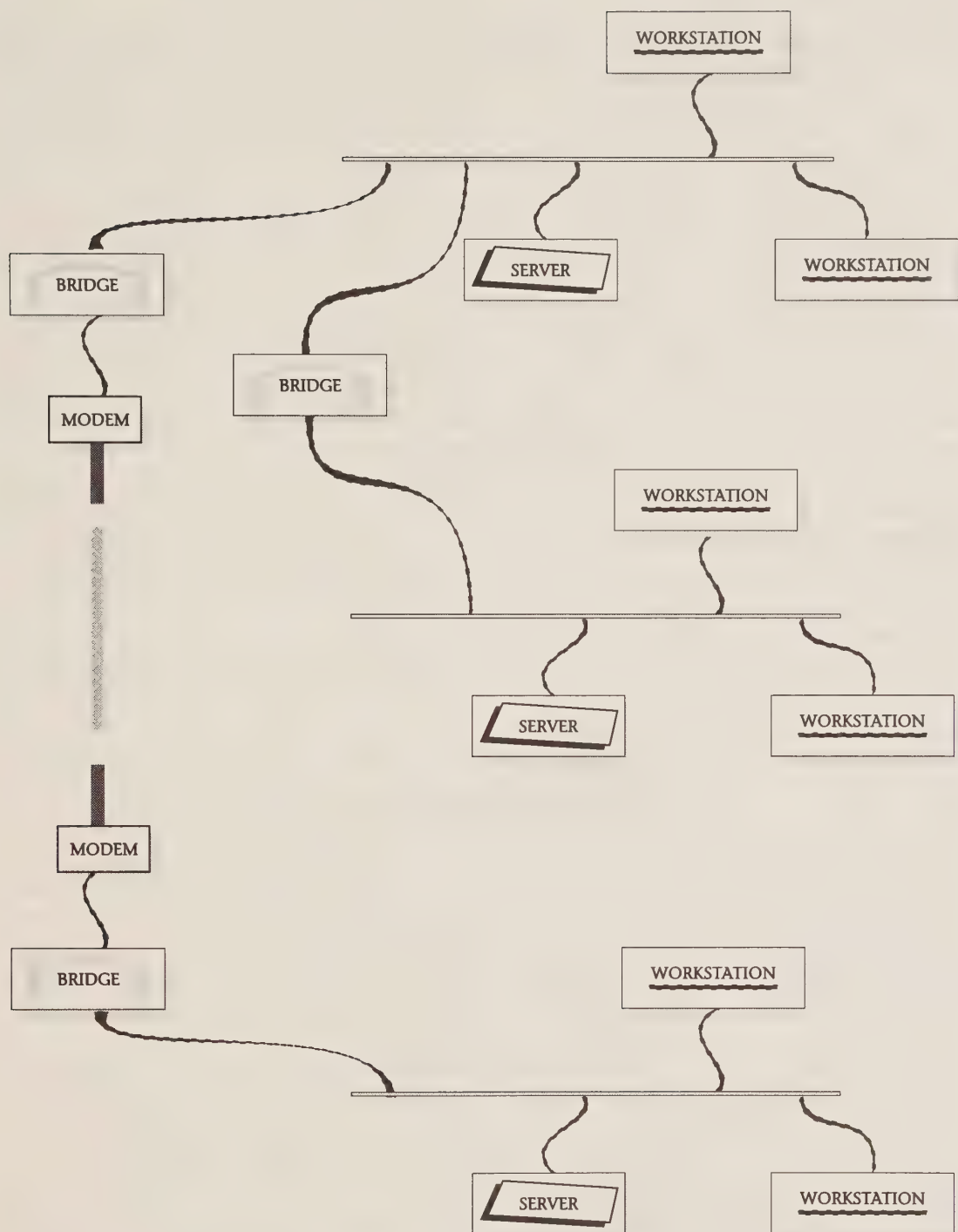
When a bridge sees an unknown destination address, however, it always “floods” the packet: sends it out every port that it has (this model assumes the bridge has multiple ports, although the typical Ethernet bridge only has two ports). After a while, the destination of the flooded packet will send a response. That response will have that node’s address on the source, allowing the bridge to update its forwarding tables to contain the location of the target node. Examination of the source portion of the Ethernet packet lets the bridge learn the location of nodes. The model assumes that a node is always on one side or the other of a bridge. If there are loops in the bridge topology, however, this model fails.

To ensure that there are not loops, bridges use a spanning tree algorithm which ensures that there is only a spanning tree and not a general mesh structure connecting the Ethernets. If two bridges would form a loop, one of the bridges is disabled. Some bridges can put themselves in a “hot” backup state, taking over if the active bridge fails. Of course, loops in a topology are not generally an issue as the job of the network manager is to structure the Ethernets into a tree structure so that this problem does not occur.

It is possible that there are several bridges in the path between two nodes (see Fig. 2-9). It is possible, in fact, to have the entire internetwork constructed using bridges. As long as there are no loops, the bridges will continue to forward packets and the entire extended Ethernet will look like one large subnetwork to the user. The problem with this approach is the rather simplistic routing method used by the bridge. We will see that the network layer uses a much more sophisticated method to keep track of multiple paths between different nodes, adjusting packet routes based on the relative performance of different paths.

Many bridges include a filtering capability that allows only certain types of frames to be forwarded. Interactive Telnet traffic might go through one set of bridges, for example, while file transfers would go through a router or another bridge.

In addition to filtering by protocol type, many environments will usually filter out broadcasts. Broadcasts may require an answer from every node. In a multisegment Ethernet, this might involve a few hundred nodes. In an extended Ethernet, this might involve several thousand nodes. If the broadcast results in all nodes broadcasting their answers, a storm of data can quickly incapacitate the network.



2-9 An Extended Ethernet

## FDDI and Token Ring

Ethernet operates at 10 Mbps. After the overhead of the data link layer and the collision avoidance mechanism, it is rare to see sustained throughput rates of more than 5 Mbps. The Fiber Distributed Data Interface (FDDI) operates at a rate of 100 Mbps, and can theoretically achieve throughput rates of 80 Mbps.

The basic FDDI configuration is a ring (see Fig. 2-10). Each station on the ring copies incoming data out to the next station. The receiver on the ring (see Fig. 2-11), in addition to copying the data through, may make a copy of the data for its own buffers.

FDDI controllers typically also include a bypass function, used when the station is not active. The bypass function allows the ring to keep operating despite the missing link. Note that the bypass function means that the signal is not being regenerated and is thus attenuating. If too many stations are in bypass mode, the signal will become too weak.

The basic FDDI operation is the same as the IEEE 802.5 token ring, which operates at 4 or 16 Mbps. FDDI, however, usually consists of two rings. The primary ring is used during normal operation. If there is a break in the primary ring, the secondary ring is used to route data back the other way. The secondary ring provides a degree of fault tolerance, providing an alternate path around breaks in the first ring.

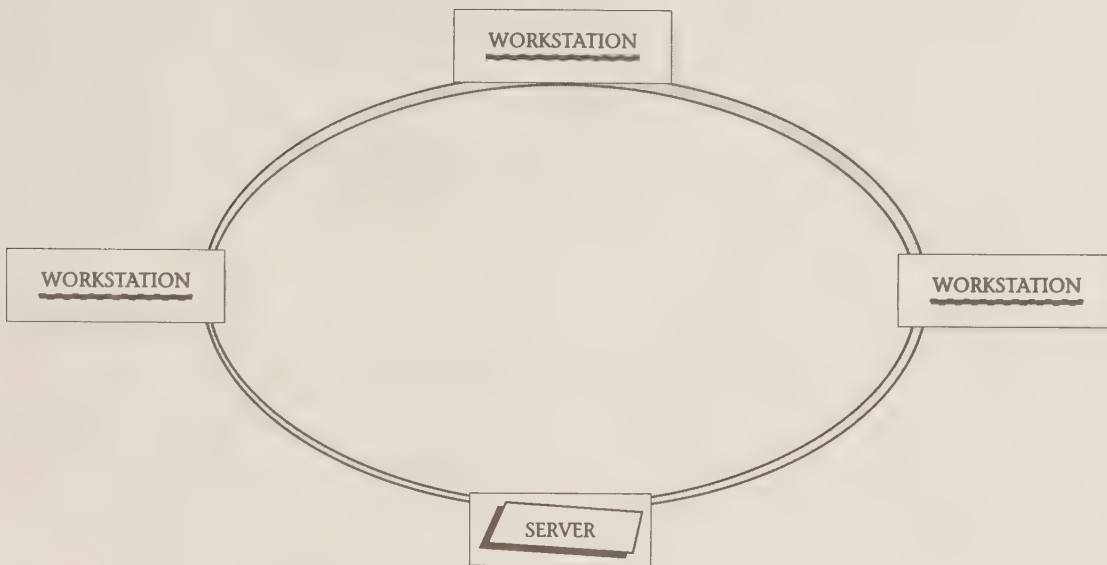
An FDDI configuration can have up to 1000 physical links on a total fiber path of 200 kilometers (km). If dual rings are being used, this is equivalent to 500 nodes on a 100-km (dual) ring.

The FDDI limits are based on a single parameter, the maximum ring latency, which is 1.617 milliseconds (ms). The maximum ring latency is the time it takes for a starting delimiter to circulate the ring. From this parameter, the total number of nodes, the maximum distance, and a variety of other ring parameters can be derived. For example, assume we have a total path length of 200 km. This introduces, at the speed of 5085 nanoseconds per kilometer (ns/km), a latency of 1.017 ms. If there are 1000 physical connections, with a latency of 600 ns, this introduces a further latency of 0.6 ms. Thus, the total latency of a 200-km 1000-connection ring is 1.617 ms, equal to the maximum ring latency parameter.

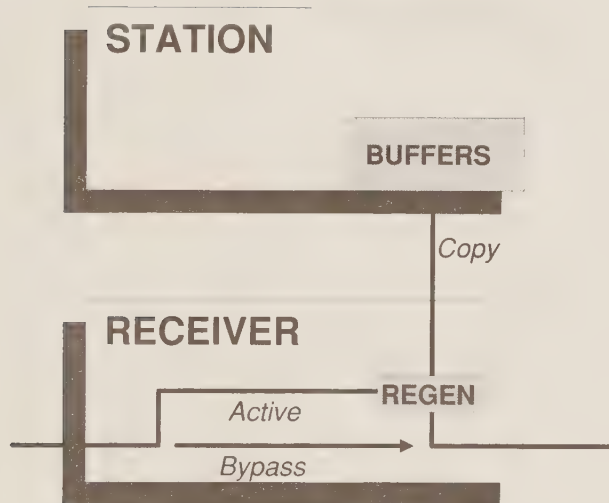
While it is possible to put nodes directly on an FDDI ring, many vendors use a multistation access unit (MAU) as a concentrator. The MAU is directly on the dual ring. Coming out of the MAU are (typically) 8 or 16 ports. If a station is inoperative, there is no problem with the signal attenuating since the bypass function is provided at the MAU.

One big advantage of the concentrator is that it is cheaper to provide single-ring controllers for workstations and servers than it is to provide a dual-ring attachment. The specification of FDDI defines both dual ring connect stations (Class A) and single ring connect stations (Class B). Figure 2-12





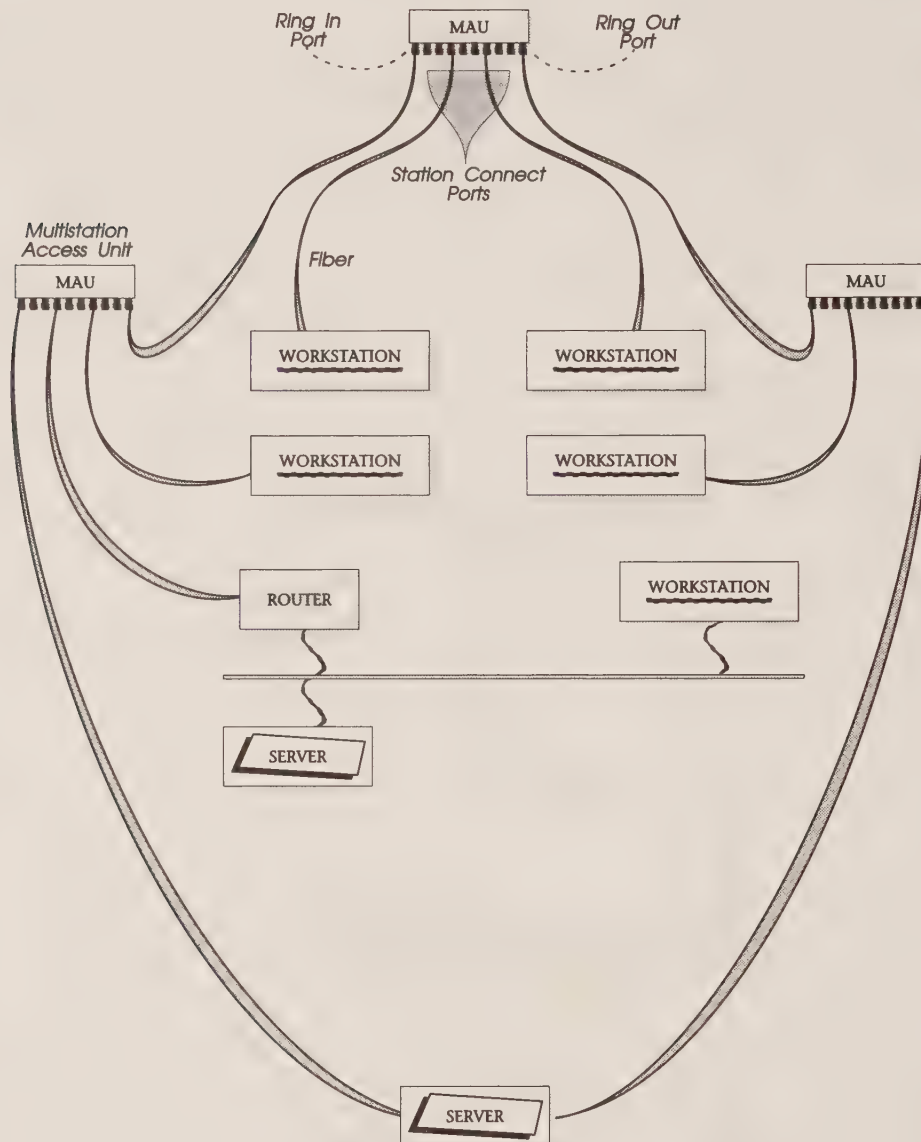
2-10 A (Very) Simple Ring



2-11 Connection to Ring

shows a possible configuration for a concentrator-based ring. Notice that a few stations are directly attached to the main ring. Most stations, however, are attached to the MAU.

Some of the stations are even further removed from the ring, being attached to an Ethernet. The Ethernet is connected to the MAU using a



2-12 An FDDI Ring

bridge. Instead of a bridge, a router could have been used. In the case of a typical bridge, the device strips the Ethernet header from the packet and puts on an FDDI header and retransmits it. This translating bridge is in contrast to the encapsulating bridge, which merely adds an FDDI header on top of the Ethernet header. The encapsulating bridge only works if the data is destined for an Ethernet, whereas the translating bridge will work on a variety of MAC technologies.

Preamble		16 or More Idle Symbols
Starting Delimiter		
Frame Control		Restricted Token?

**2-13**

An FDDI Token

### *FDDI Operation*

In an Ethernet, any node can transmit if the medium is free. In a token ring such as FDDI, a node can only transmit when it receives a special packet, called the token (see Fig. 2-13). The token is passed from node to node on the ring. If a node has nothing to transmit, it merely copies the token through. If it does have something to transmit, it captures the token. The capture process consists of failing to copy the token and instead substituting idle symbols.

The node then has a set amount of time, governed by the token holding timer (THT) before it must replace the token on the ring. During that time, the node is free to send one or more data frames. A restricted token allows two (or more) nodes to engage in a dialogue by passing the token back and forth to each other without having to worry about other nodes cutting in.

Once a data frame is sent, it will circulate through the ring. Presumably, at some point a node will see its address and copy the data into its buffers in addition to copying the data through to the next station. When the frame goes back to the sending node, it will strip the frame from the ring by replacing the data with idle symbols.

Figure 2-14 shows the format of the data frame. At the end of the frame is the frame status field, which is used to see if a destination node recognized its address. If it did, it flipped this bit. If it also had enough room in its buffers to copy the frame, it flipped the frame copied bit. The error detected bit is used if a frame check sequence fails.

The data field is theoretically limited to roughly 4500 bytes, although most users of the FDDI will have much smaller packet sizes. Inside this data field would be the LLC header, followed by the headers for upper layers.

The frame control field is used to govern the basic operation of the ring. The address length bit, for example, selects either a 2-byte or the standard 6-byte address. The frame type indicates if this is an internal MAC-management frame, one used for the FDDI station management (SMT) function, or is an LLC data frame. More information on the frame type, priority bits, and the class bit are contained in the following sections.

Preamble		16 or More Idle Symbols
Starting Delimiter		
Frame Control		Class Bit: Synch/Asynch
	Address Length Bit	
	Frame Type	
	00 - MAC/SMT	
	01 - LLC	
	10 - Implementor	
	11 - Future Use	
	Priority Bits (LLC Frame)	
Destination Address		
Source Address		
Data		
Frame Check Sequence		
Ending Delimiter		
Frame Status Field		Address Recognized?
		Error Detected?
		Frame Copied?

**2-14**  
An FDDI Frame

### *Modes of Operation*

An FDDI ring operates in two modes:

- Synchronous mode guarantees a certain amount of bandwidth and response time to nodes.
- Asynchronous mode provides dynamic bandwidth sharing.

Asynchronous mode is instantaneously allocated via the token, while synchronous bandwidth is allocated ahead of time using SMT protocols. Note that initial FDDI implementations do not make use of synchronous mode.

Synchronous bandwidth is allocated as a percentage of the target token rotation time (TTRT), that is a portion of the total bandwidth on the net-



work. Needless to say, the sum of the synchronous allocations should be less than or equal to 100 percent.

Each node with a nonzero bandwidth is allowed to transmit data frames for a period of time without having to worry about starting the token rotation timer. If, after all nodes are done with their synchronous transmissions, there is still some bandwidth available, nodes can do asynchronous transmission up to the limit of the token holding timer.

Asynchronous transmission is a two-tier allocation of the bandwidth:

- Nonrestricted mode provides time-slicing among all nodes that wish to send data.
- Restricted mode is dedicated to a single extended dialogue.

The normal operation is nonrestricted mode. In this mode, allocation of the token is based on a priority scheme, which is based on the amount of time it takes for the token to circulate the ring. Each priority level has a threshold token rotation time (TRT).

As the TRT gets longer and longer, the lower priority levels are cut off. Nodes can then only send data of higher priorities. As the token goes around the ring with a high-priority, eventually all nodes will have sent their high-priority data. This means that the token will go around the ring more quickly, allowing lower-priority data frames to be transmitted.

The target token rotation timer is the total bandwidth available on the ring. After the synchronous transmission is finished, this leaves a certain amount of remaining bandwidth, symbolized by the token rotation timer. The difference between the current TRT and the target TRT (the TTRT) is thus the available asynchronous bandwidth—the minimum value of the TRT is equivalent to nobody sending synchronous traffic.

Restricted mode is entered by two nodes when a nonrestricted token is received. The first node sends an initial batch of data and then issues a restricted token. The receiving node sends its data and then sends the restricted token back out the ring. Restricted mode prevents all unrestricted asynchronous traffic (including basic station management tasks such as exchanging neighbor IDs). The decision to enter, terminate, or continue a restricted dialogue is up to the higher layers that are using FDDI. Since synchronous transmission is unaffected by the token, it is unaffected by the restricted mode.

One of the functions of the station management is to negotiate a maximum restricted mode time. Note that in restricted mode, there is really no need to obey the token holding timer—by its very nature restricted mode has already preempted fairness with other nodes.

The use of restricted and synchronous mode are optional features in FDDI. Restricted mode would presumably be used by critical functions, such as system management. Synchronous mode would be used by real-

time applications, such as voice, or time-critical applications, such as remote booting of nodes.

### *Ring Initialization and Recovery*

Any station that detects a need to initialize the ring issues a claim token frame. Multiple stations bid for the right to lead the initialization of the ring by continuously transmitting claim frames.

Each station then looks at an incoming claim frame to detect if the frame is the one originally sent or another station's bid. Each frame contains a bid value, and the highest bid wins.

In the case of conflicting highest bids, arbitration is based on target token rotation time (TTRT), which is used to indicate how long a token should take to circulate the ring continuously. The lowest TTRT wins the bidding process. If there are multiple low TTRT values, the highest station address wins.

As soon as a station wins the claim token process, it begins the ring initialization process. First, it sets the target token rotation timer to the value in the successful claim frame. It then resets the token rotation timer (used to detect a token that has gone awry) and issues a nonrestricted token.

This token will go through the ring three times before normal data traffic is possible. On the first round, each station sets the TTRT value and then sets an internal flag to indicate that the ring is operational again. On the second round, synchronous traffic is issued. On the third round, asynchronous traffic can begin.

The claim token process is timed by each station. If the timer expires and the bidding is unresolved, the station begins transmitting a beacon. Note that only stations that are still in the bidding process can send a beacon. If a station is out of bidding and its timer expires, it enters the bidding process again.

The purpose of the beacon frame is to try to pinpoint where in the ring a problem is occurring. The beacon says that a significant logical break in the ring has occurred, either because a station is inoperative or because the medium has a break. Each ring transmitting beacons will transmit them continuously. As in the case of the claim token, any incoming beacons are examined. If the station address on the incoming beacon is different, that means that there is a station upstream also sending the beacon and the node stops transmitting.

As each node yields, the source of the beacon gets closer and closer to the break in the ring. At some point, when the ring is fixed, a node will see its own station address in a beacon. This means that the ring is operational again, and the claim token process is entered.

## Token Ring Support

Token ring networks are not used as a subnetwork method for connecting together Sun workstations, but the token ring can be integrated into the network. This is important since many IBM networks, both PCs and larger systems, are based on the token ring subnetwork.

Token ring systems environments can be running one of three different types of networks. First, they can be running IBM's SNA. If this is the case, the TCP/IP world is connected to the SNA world via a gateway. A second option is for the token ring to be using some PC network, such as Novell, Banyan, or LAN Manager. Most PC networks have a TCP/IP support. Some, such as Novell, have explicit support for protocols all the way through NFS. The third option is to simply run TCP/IP on the whole network. PC-NFS and various IBM minicomputer and mainframe software packages allow all the protocols discussed in this book to be used.

An easy way to connect the token ring to a broader TCP/IP-based network, no matter what protocols the token ring is running, is by using a gateway system. The system, like any gateway, has at least two adapters, one for each network. The Sun product to do this is the SunNet TRI/S adapter, which provides token ring support for the S Bus. There are a variety of other products that connect Ethernet to token ring networks, either directly or over long distances.

Other systems can also be used as gateways. For example, to use a Sun-3 or Sun-4, the Formation fv1600 is a VME bus interface that supports both 4- and 16-Mbps token rings. PC systems can also be used as gateways.

## Wide-Area Links

Ethernet and FDDI are the primary LAN technologies for Sun networks. There are a few others, such as HYPERchannel, Ultranet, and HIPPI links, which will be shown in the case studies at the end of this chapter. Now, however, a few wide-area technologies will be examined.

There are two classes of protocols that are used in a wide-area environment. First, there are formal wide-area data networks, based on protocols like X.25. Other multiple-access data networks are Frame Relay, SMDS, ISDN, and similar protocols. The other class of protocols is used for dedicated point-to-point links. These protocols are much simpler because there are only two computers communicating, so the concept of virtual circuits and addressing is not necessary.

There are two primary protocols that are used for TCP/IP networks. The simplest one is SLIP, the Serial Line IP. A more modern successor to that is the Point-to-Point Protocol, PPP.



## *SLIP*

The Serial Line Internet Protocol is known as a “nonstandard”: everybody uses it, but it has remained a de facto standard. This is because it so simple that it doesn’t really need much to make it work. The typical use of SLIP is a modem connection from 1200 bps to 19.2 kbps.

SLIP comes from 3Com’s UNET TCP/IP implementations. SLIP is just a packet framing protocol: no addressing, no packet type identification, no error detection or correction, just data. Because SLIP was added to Unix 4.2BSD by Rick Adams and is present on Sun workstations, it quickly caught on as a way to connect hosts to remote routers. As well as being a standard part of 4.3BSD, it is present in Ultrix, SunOS, terminal servers, and IBM PCs.

SLIP has two special characters: end and escape (END) and (ESC). A simple character stuffing method is used to make these control characters transparent from the normal data stream. If you are going to put an END character in as data, you substitute ESC plus octal 334. If you are going to send ESC in the data, you send ESC plus octal 335 instead.

To send a packet, you simply send data plus the END character. Note that a simple upwardly-compatible extension, suggested by Phil Karn, is to start and end with an END character. This flushes any erroneous line noise bytes.

If there is no line noise to flush, a node will get two ENDs back-to-back. This looks like a zero-length packet for IP, which will simply throw it away. The SLIP implementation could also do the throwing away.

SLIP implementations, being de facto implementations, may or may not work together. Usually, the only thing that would stop them working, however, is a difference of opinion about the size of the maximum packet size. The Berkeley implementation handles 1006 bytes and functions as a sort of de facto maximum transfer unit (MTU) for the de facto standard.

## *PPP*

SLIP has some real limitations, not the least of which is a single user per line. There are no provisions for addresses for different upper-layer users. A more modern alternative to SLIP is the Point-to-Point Protocol (PPP). Work on this protocol started in 1988 and by 1990 it began to have interoperability demonstrations.

PPP works over any DTE/DCE physical interface (a term for the physical link between a terminal and the device that hooks to the communications network), including popular standards such as RS-232-C, RS-422, and CCITT V.35. The only requirement is that a duplex circuit is needed. There is no requirement for control signals, but they certainly make life easier when doing an implementation.



At the data link layer, PPP works on synchronous and asynchronous circuits. Framing is very similar to the ISO High-Level Data Link Control (HDLC) protocol. Since HDLC framing is an international standard, it is often done in hardware: PPP describes which values to use in the standard HDLC fields and then has an additional protocol field. There are three groups of protocols: link control, network-layer protocol control, and network-layer protocol data.

PPP sits in between the IP layer and the underlying physical layer. During link operation it plays a very simple role: each PPP packet contains an IP packet. By providing a standard encapsulation, however, the TCP/IP software is insulated from the details of underlying physical links.

Establishing a PPP link involves four steps. First, the link is established and an appropriate configuration negotiation established. Options that are negotiable include:

- Maximum receive size
- Async control character mapping
- Authentication method
- Encryption method
- Link quality monitor parameters

After configuration, there is an optional authentication phase. This can be a simple user/password protocol or the always-popular “none.” Authentication is important since PPP is a key protocol for dial-up IP networks.

Next, there is an optional link quality determination. PPP doesn’t specify how a node makes the policy decision that a line is not acceptable. Instead, it provides echo requests and replies to allow various policies to be used (e.g., error rate or throughput).

Link monitoring is based on a way of determining how many packets and octets are lost. Each end counts the numbers of packets and octets sent and received and then makes that information available to the other side. The trick is to synchronize the state of this data between the two sides.

This synchronization is done using a Link Quality Monitoring (LQM) packet as a checkpoint. You send an LQM along with your numbers. When the other side receives the counter, it checks with its version. By comparing the two, you can arrive at the number lost.

Finally, the link enters the link ready state where network-layer protocols are set up. The IP Control Protocol (IPCP) is the protocol used to enable and disable IP. Options for negotiation here include the IP address and the compression types.

The IP address option allows a node to verify or assign an IP address. The field has two subfields: My-IP-Address and Your-IP-Address. When a PC dials into an IP network, it will not know its IP address since that address will change depending on the particular port the node is on. This option

allows the remote node to determine what address it has for the purpose of this session.

Compression allows the header compression method used by Van Jacobson to be used. Think of an X Windows or Telnet session coming over a dial-up line. A large portion of each packet consists of the IP and TCP header. The Van Jacobson header compression gives a higher payload to each packet, thus giving much higher effective throughput on low-speed lines.

## X.25

At the data link layer, the X.25 protocols are a subset of HDLC. In addition, X.25 defines functions at the physical and the network layer. The network layer function of X.25 is used to set up a virtual connection between two nodes in the X.25 subnetwork. Then, the LAPB protocols define how a user of X.25 interfaces with the network. X.25 is actually used in a wide variety of different portions of the network.

X.25 is connection-oriented. The user at the network layer sets up a virtual circuit to a remote destination. The virtual circuit can stay permanently in place or can be set up dynamically (a switched virtual circuit).

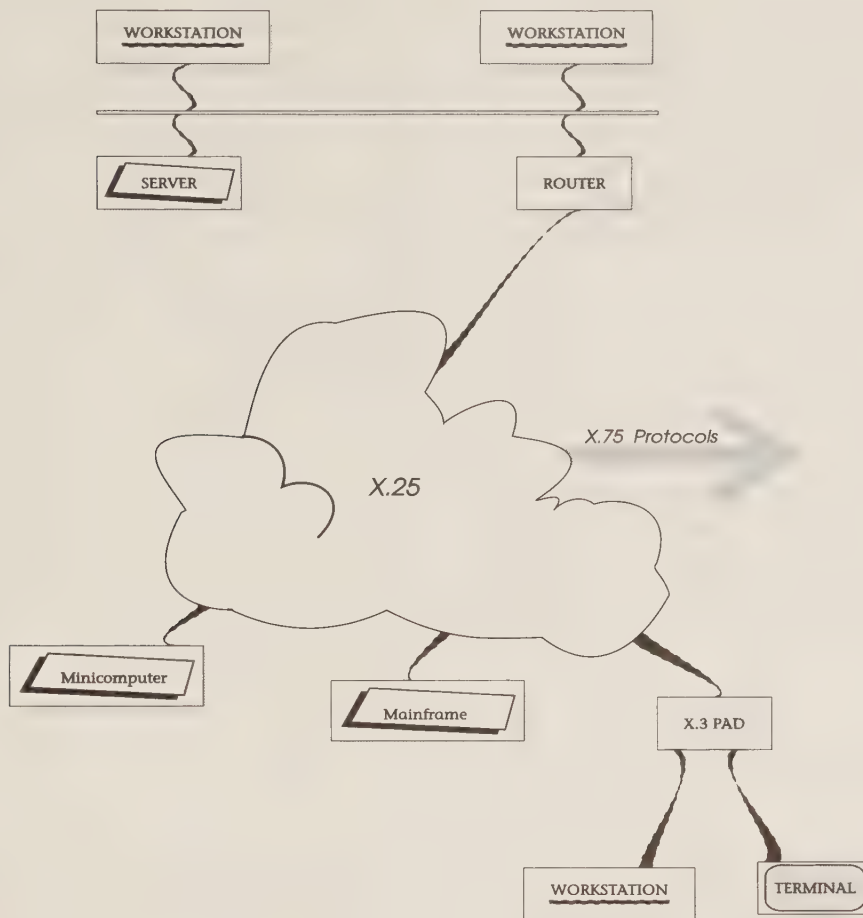
Figure 2-15 shows a typical X.25 configuration. Notice that X.25 only specifies the interface to the network: the inner workings are implementation-dependent and are thus represented as a cloud.

X.25 can be used for a wide variety of purposes. For host-to-host communication, a higher-level program is resident on both computers. Thus, the router on the Ethernet might set up a virtual circuit to a minicomputer or mainframe or even another router. We will see in the next chapter that X.25 can be used as a means of connecting multiple LANs together.

X.25 is also a means for a terminal to communicate with a host. The X.25 protocols define synchronous data transmission, but terminals are usually asynchronous. A device known as a packet assembler/disassembler (PAD) is used to interface the asynchronous device to the synchronous X.25 network. The operation of the PAD is defined in the X.3 CCITT standard.

Two additional protocols are used to define how the terminal (or the workstation emulating a terminal) and the host communicate with each other. The X.28 protocols define how the terminal interfaces to the PAD. X.28 defines things like when the PAD should take data received and bundle it up in an X.25 packet. The X.29 protocols define how the host can communicate with the PAD and thus control the terminal.

X.25 access can be provided in a variety of ways. Usually, a router is configured with both an Ethernet board and an X.25 interface, as in the case of SunLink X.25 or any other gateway system. Most of these gateways allow both terminal use with X.28 and X.29 and use by the network layer, IP.



2-15 An X.25 Configuration

X.25 interfaces are important in a variety of situations. Any private network that has a variety of destinations, as in the case of bank branches, may want to use a set of leased lines and provide an X.25 network over that. Alternatively, public X.25 networks are used for occasional access. Once connected to one X.25 network, it is possible to reach a destination target on another X.25 network. The X.75 protocols define how one X.25 network interconnects to another. A switched virtual circuit could thus be established across national boundaries.

### SMDS

The Switched Multi-megabit Data Service (SMDS) is a very high-speed network, often targeted for metropolitan area networks (MANs). SMDS is also offered as a WAN service by the long distance carriers. SMDS can operate



at various speeds, but most initial trials operate at DS-1 (1.544 Mbps) or DS-3 (44.736 Mbps). Eventually, SMDS will use Synchronous Optical Network (SONET) transmission rates up to 150 Mbps and higher.

The SMDS standard is virtually identical to the IEEE 802.6 MAN standards. Both include provisions for voice and video, but initial prototypes operate over just data. The basic idea is to allow a user access of T1 to T3 speeds on demand: SMDS is a switched service.

SMDS is based on the fact that many applications are bursty: their data requirements vary widely over time. This is not always the case: voice, for example, is fairly constant in its bandwidth needs.

In an environment with many different applications (i.e., TCP/IP), the requirements are harder to measure. Traffic may consist of a few telnet sessions one minute, and a multi megabyte data file transfer the next minute.

SMDS is being positioned as a precursor to Broadband ISDN (BISDN). BISDN will offer bandwidth in multiples 45 Mbps, with the basic channels operating at 150 Mbps or 600 Mbps. SMDS is compatible with (but not dependent on) SONET: a fiber optic transmission standard with rates ranging from 51.84 Mbps up to more than 13 Gbps. There are estimates that over 60 percent of the interexchange network traffic will be converted to SONET.

### SWAN: An Example

There are many different internetworks, many of which are connected to the Internet, an international TCP (and OSI) internetwork. To see how collections of networks are all collected together, let's begin with Sun's own corporate network, the Sun Wide-Area Network (SWAN).

SWAN has more than 10,000 nodes in 20 countries, all based on TCP/IP protocols. Although TCP/IP is the main protocol suite, DECnet, Open Systems Interconnection (OSI), and the Systems Network Architecture (SNA) also play a role.

In the United States, a double T1 network (a T1 line is a 1.544 Mbps leased digital line) forms the core of the network (see Fig. 2-16). This T1 bandwidth is allocated using T1 multiplexers for voice, video, and data. Fifty-six kbps links are used to connect to the United Kingdom, France, and Japan.

Within the San Francisco Bay Area, the core of the Sun network, a DS3 (45 Mbps) fiber link connects the main sites in Mount View, Milpitas, and Palo Alto. Two different Internet gateways provide access to the outside world.

Each of the hub cities serves as a distribution point for their area. Figure 2-17, for example, shows the western area, whose hub is in Milpitas, California, near San Jose. Milpitas and Billerica are both crucial hubs for Sun.





2-16 The SWAN Backbone

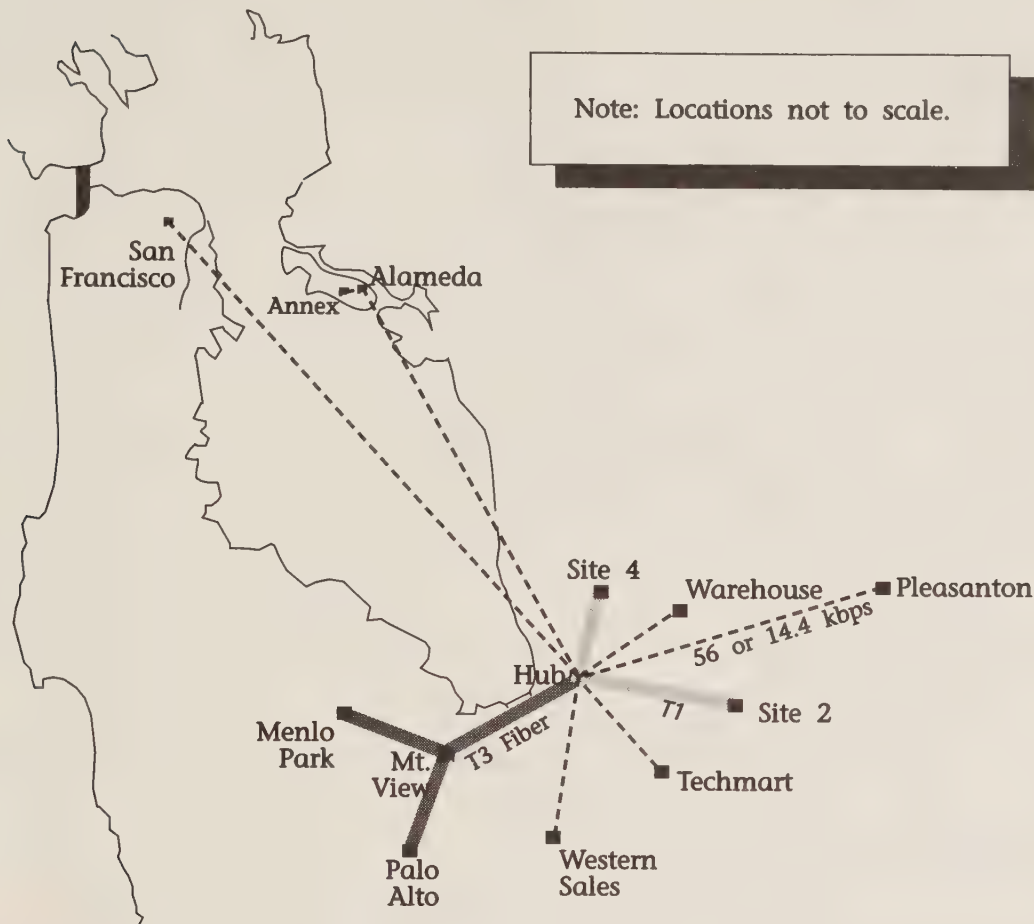


2-17 SWAN Western States Area

Billerica is the gateway to Europe, with leased lines and satellites providing the link to the European hub in England.

Milpitas serves as the gateway to the Far East but is also the hub for the Bay Area, where much of Sun's facilities are located (see Fig. 2-18). Milpitas is connected to the other major corporate campuses in Menlo Park, Mountain View, and Palo Alto using fiber running at T3 speeds. Other area facilities are linked with T1 or slower lines.

The entire SWAN backbone is available for a variety of applications, ranging from voice to video to (of course) data. T1 multiplexers are used to concentrate the different types of traffic together. Figure 2-19 shows a typical configuration for linking the computer equipment to the SWAN.



2-18 SWAN Bay Area Backbone

Two Sun-4/390 systems are used to route all data between the Milpitas and Billerica locations. Each hub has a SWAN backbone, currently an Ethernet system. No users are on these backbones, only routers.

Milpitas, for example, has a router that connects the SWAN backbone with the Milpitas facility backbone. That backbone in turn connects to smaller Ethernets in various buildings or laboratories.

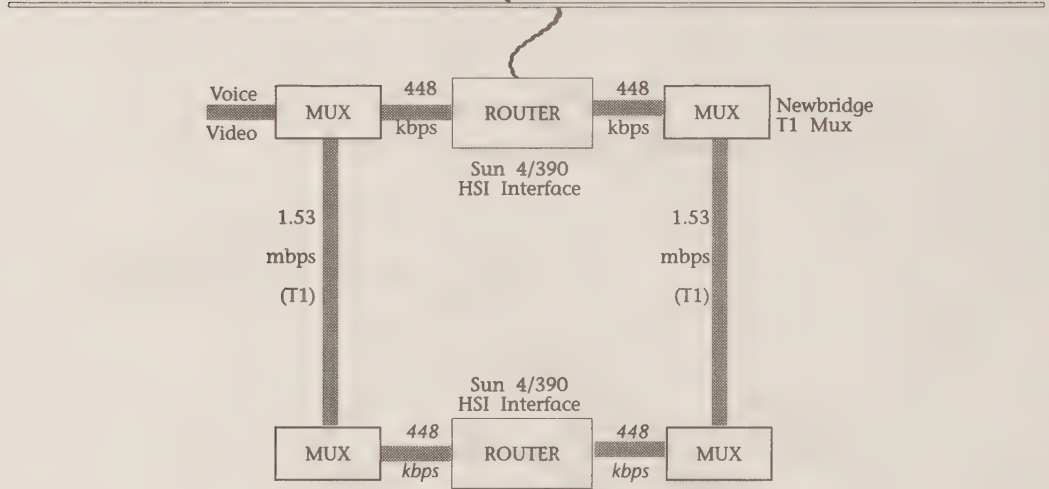
Segmenting traffic this way is done for both performance and security. Traffic that is coming from Europe and going to Houston, Texas, would stay on the SWAN backbone. Only traffic that would end up at one of the Milpitas buildings would make it through the router to the Milpitas backbone.

The main SWAN link between Milpitas and Billerica uses two pieces of equipment besides the server. The T1 multiplexer is used to concentrate

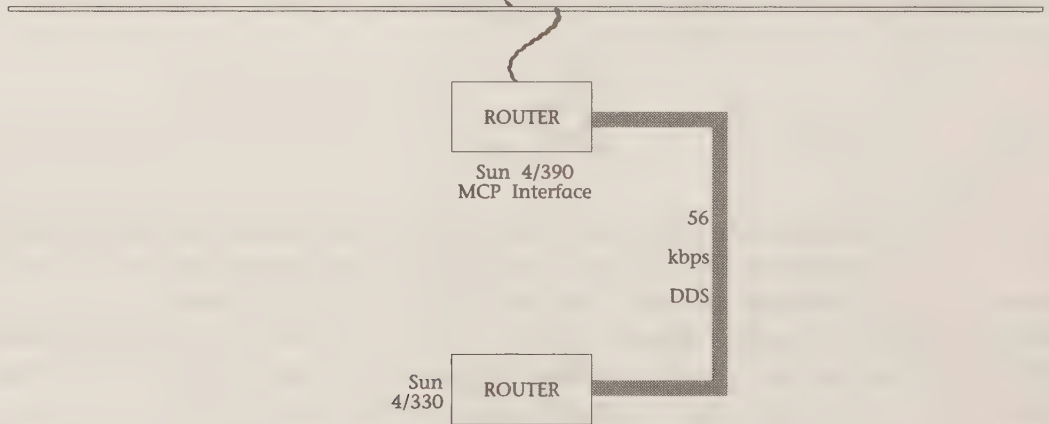
Milpitas Facility Backbone



Milpitas Wide-Area (SWAN) Backbone



Billerica Wide-Area (SWAN) Backbone



New York Sales Office

2-19 Data Links to the SWAN Backbone



different types of traffic together. The board on the Sun that drives the data portion of the link is an High Speed Serial Interface (HSI) communications controller, in this case providing links running at 448 kbps, a portion of a T1 link between the sites.

Between Billerica and local sales offices there is no need for that level of performance, so a cheaper Multiprotocol Communications Processor (MCP) board is used to drive the 56 kbps links over a digital leased line. A router in New York is connected to the office backbone, to which a series of workstations and other servers would be attached.

This network allows any user at Sun to communicate with any other user. The network handles a very large amount of traffic. Over 10,000 corporate users are on the system, and there are links to 20 countries. Three different Internet gateways provide access to the Internet, plus there are many UUCP-based links.

The most notable form of outside access is the external mail relay host, which can easily handle 60,000 external messages per day. That mail hub moves the data over to subdomain mail hubs that distribute the messages to individual users.

Most mail distribution is done internally with the Simple Mail Transport Protocol (SMTP) over TCP. It uses the Domain Name System (DNS) to find users. In addition, there is an X.400 link to connect to X.400-based message handling domains. A Fax Mail capability allows a user to send fax messages using a window-based fax tool, a utility very similar to the mail tool.

The network makes extensive use of NFS and the Automounter to make files available. Many users have their window system remotely mounted even if they have a local disk drive.

Mainframe access is provided using two different Sun Channel Gateways. It is interesting that all access to the IBM mainframes is done via Sun workstations: no 3270 terminals are used. Much of the mainframe work is access to a Cullinet DBMS for order entry and shipping status. Other mainframe applications are MRP (a production materials planning technique) and accounts payable systems. An average of 500 to 600 users are on the system.

NFS is extensively used, including mounting of MVS data on Sun workstations. Over 1000 large servers are used in the network. Engineering groups in the eastern United States and in Europe use transcontinental mounts to access source systems located in the western United States. The key to the wide-area NFS access is having sufficient bandwidth, in this case provided by dual T1 links.

A few servers are available to thousands of users. For example, there is a single bug database server machine accessible to the entire corporation. Another example is the news server, where one machine provides news access to the entire engineering group.

Names are organized around several different NIS domains. Several NIS servers handle more than one domain. All names, passwords, and aliases are stored on a single corporation-wide server. The maps are then pushed nightly to all domain servers. Note that maps are now very large: some are greater than 1 Mbyte. The `ypxfer` utility, discussed, in Chapter 9, is used to transfer information from the NIS manager to groups.

Source code machines are all specified on global Automounter maps, maintained with the NIS naming service. A user can change to any other user's home directory by using a corporation-wide Automounter map, assuming, of course, the user has the appropriate NFS permissions.

A variety of RPC-based programs are available in addition to NFS. For example, the network has a lights database. All cars in the corporation have their license numbers entered. If you see a car with its headlights on, typing "lights" and the license number will send a mail message to the owner of the vehicle.

Possibly the single most popular RPC program at Sun is the quote server, which gives current stock information for Sun stock (as well as leading competitors). Since most executives have a stock package, this is often the first program they learn to use. Even senior engineers will often keep the Stock Tool up next to their debugger, octal dumps, and vi editor.

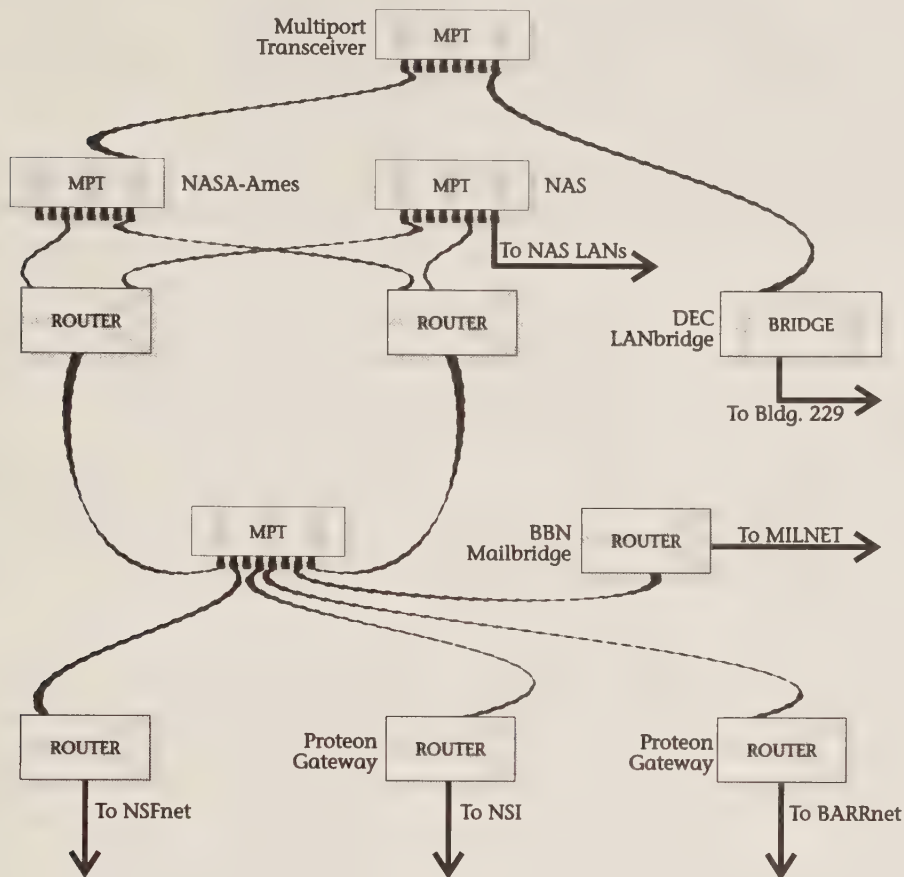
Management of this network is broken up across domains. Individuals (or groups) are responsible for their department's computers. A central Information Resources (IR) group handles the central facilities and SWAN. SunNet Manager is often used for managing this network, especially with SNMP, DECnet NICE, and IBM Channel agents to monitor individual devices.

## On the Internet

Let's say a user at Sun engineering wanted to access services located at a research facility on the network, say a Cray computer for modeling some circuit design problem. Two sample locations are described in this section and we show how Sun is just one of many corporate and research networks that are part of the Internet.

Remember the Sun Milpitas SWAN backbone, used to link the SWAN regions together? This backbone also contains routers that are connected to the Bay Area Regional Research Network (BARRnet).

An engineer in Mountain View might wish to transfer a file over to another network, where a supercomputer might compile and run the file. An appropriate protocol would be picked for this operation, such as the File Transfer Protocol.



**2-20** NASA-Ames Wide-Area Links

After some initialization, a packet would originate from the engineer's workstation with some foreign IP address. The packet would be forwarded by intermediate routers until it reached the Sun Milpitas backbone.

There, the routers connected to BARRnet would see that the Internet address was foreign and would send it out over BARRnet. One possible target location for this packet might be the Center for Numerical Aerodynamic Simulation (NAS) located at the NASA-Ames Research Center, near Mt. View.

Our packet would thus go from Sun's Mt. View facility to the SWAN backbone in Milpitas and then on to the NASA-Ames wide-area backbone. Figure 2-20 shows a portion of the configuration for that backbone.

Three major routers (or sets of routers) are available for the major networks for which NASA-Ames is a hub. The NSFnet is the core national re-



search network to which regional networks like BARRnet are connected. Most traffic going from BARRnet to NSFnet would go over this backbone.

Notice that the backbone is not even a piece of cable; it is a multiport transceiver, the “Ethernet in a Can.” This level of the network also contains links to Milnet, the military network, and to NASA’s own WAN, the NASA Science Internet (NSI).

If we had permission to use the NAS facilities, our packet would not go out through one of these routers but would continue on into the NASA-Ames network. There are two large gateways that provide access to NASA-Ames, where traffic is then routed to NAS or other research facilities.

The redundancy of routine is useful since it means that there will always be a path into the network. Those routers are in turn connected to various NASA-Ames backbones, including the one at NAS. Figure 2-21 shows the configuration once it reaches that network.

Traffic comes in off the network onto the NAS backbone Ethernet, shown on the left of the figure. The user would then go through at least one more router and reach some server system. Usually, remote users are not logged directly onto the Cray computer, the Cray being an expensive terminal server.

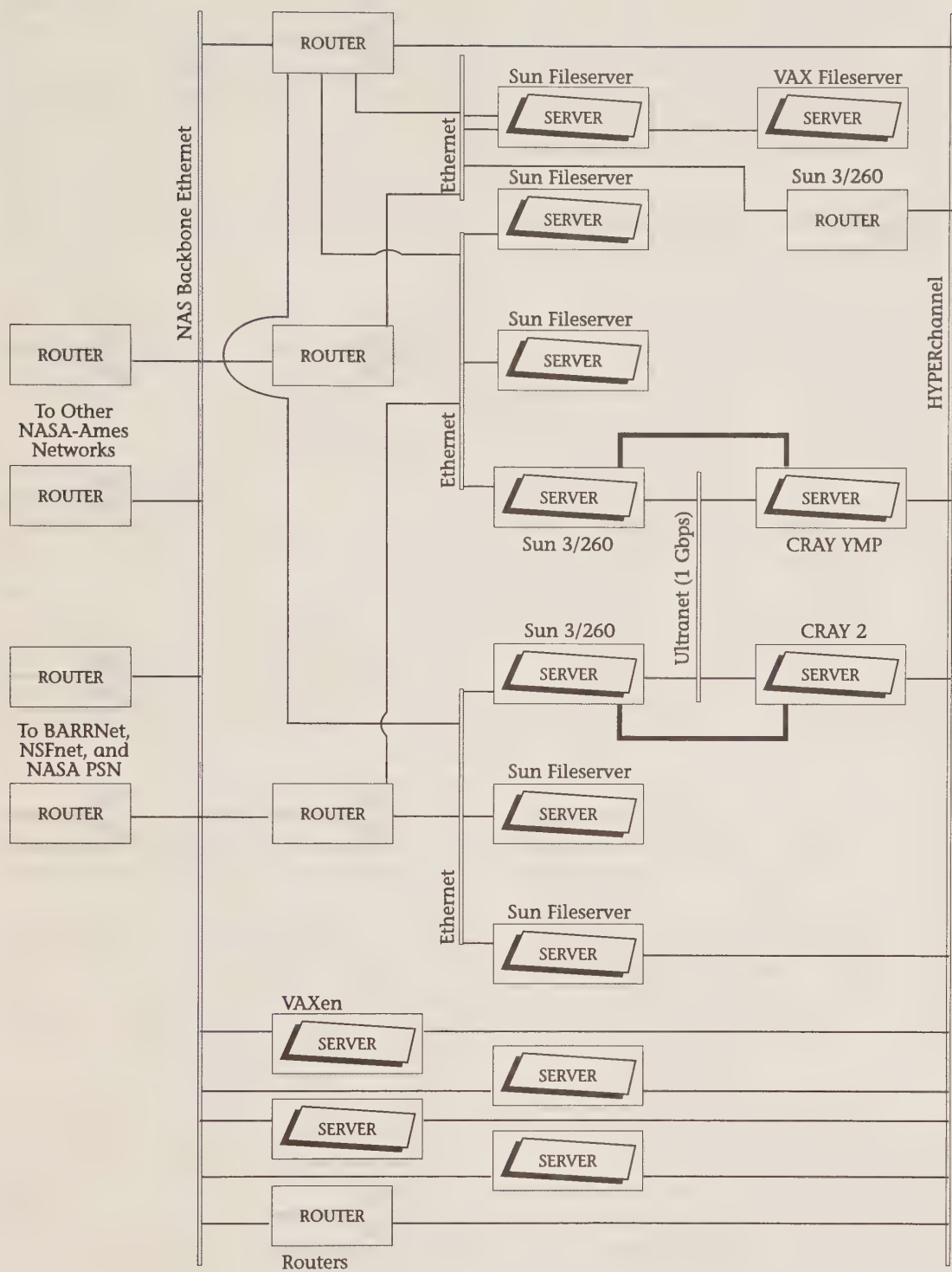
Users might use one of the VAX systems as a front-end system if they were coming in from the Internet. A Telnet session would be set up from the Mt. View workstation to the VAX at NAS. From there, additional commands might be prepared and sent to the Cray computer or some other system. At this point, our user is just like any other local user. Well ... not quite. Our user has a terminal session at this point, whereas local users are probably running X Windows to put a nice display on their screen. This is not a given, however, since even X Windows can be run in a wide-area environment, particularly if the path between the two nodes is fairly fast and if techniques are used for compressing the lower layer protocol headers.

The NAS network consists of a series of Ethernet systems to which are connected the local servers. In addition, there are many Sun and Silicon Graphics workstations that are used by local users.

Most of the server systems are connected to both an Ethernet and to the Hyperchannel. This medium-speed network is used for server-to-server communication, as in the case of mounting file systems.

Two special servers are the two Cray computer systems. Instead of connecting these systems to the Ethernet, they have a link to the Ultranet network running at 1 Gbps. This high-speed communication channel is used, among other things, as a way of cross-mounting many gigabytes of disk space between the two machines using the Network File System. There are also some dedicated Sun systems that have high-speed direct links into the large servers.





2-21 The NAS Network

Let's say that our engineer didn't have access to NASA-Ames. Instead, a commercial account is set up at one of the national supercomputer centers, most of which allot a percentage of their capacity to commercial use.

Our packet would still travel to the NASA-Ames backbone via BARRnet. Instead of being sent into the NAS network, however, our packet would be sent out on the Internet. If the account was at the San Diego Supercomputer Center (SDSC), a likely choice for a California user, the packet would go through the NSFnet backbone (at T3 speeds) and reach the SDSC backbone (see Fig. 2-22).

The SDSC is one of the hubs for the NSFnet, but, like NASA-Ames, also serves as a gateway to many other networks. A hierarchy of multiport transceivers is used to connect routers together. Many of these routers are providing T1 or 56-kbps links for DECnet or TCP/IP traffic to regional nodes. Some of these are links dedicated for supercomputer access by clients, whereas other links are used for regional networks.

SDSC serves as a hub for both NSFnet and other networks, including the CERFnet regional network, shown in Figure 2-23. Notice how each hub has a terminal server, which allows a user with a simple terminal to access services on the Internet.

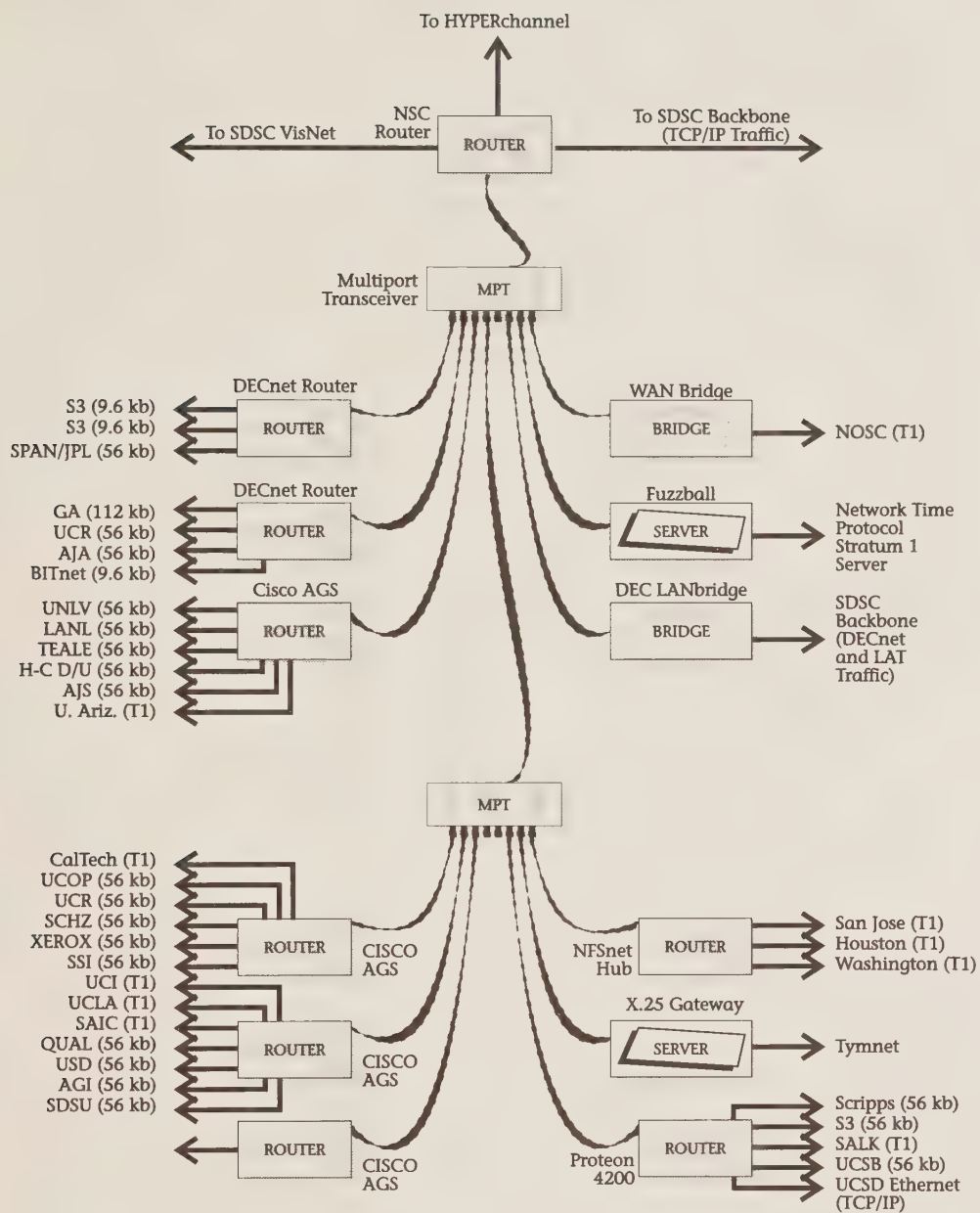
Since our user is not going out to one of the CERFnet facilities, however, the packet will continue on into the SDSC network, shown in simplified form in Figure 2-24. The WAN backbone is the entry point for remote users. Eventually, most of these remote users will go over the WAN backbone and down to the SDSC Ethernet backbone at the bottom of the diagram. There, as in the previous example, the user would set up a Telnet session with a front-end server, in this case a set of VAX minicomputers.

The VAX systems are connected together using a 70-Mbps VAX Cluster system. This provides a workspace for remote users, as well as the ability to drive specialized output devices such as a film recorder used to capture Cray-driven animation for movies.

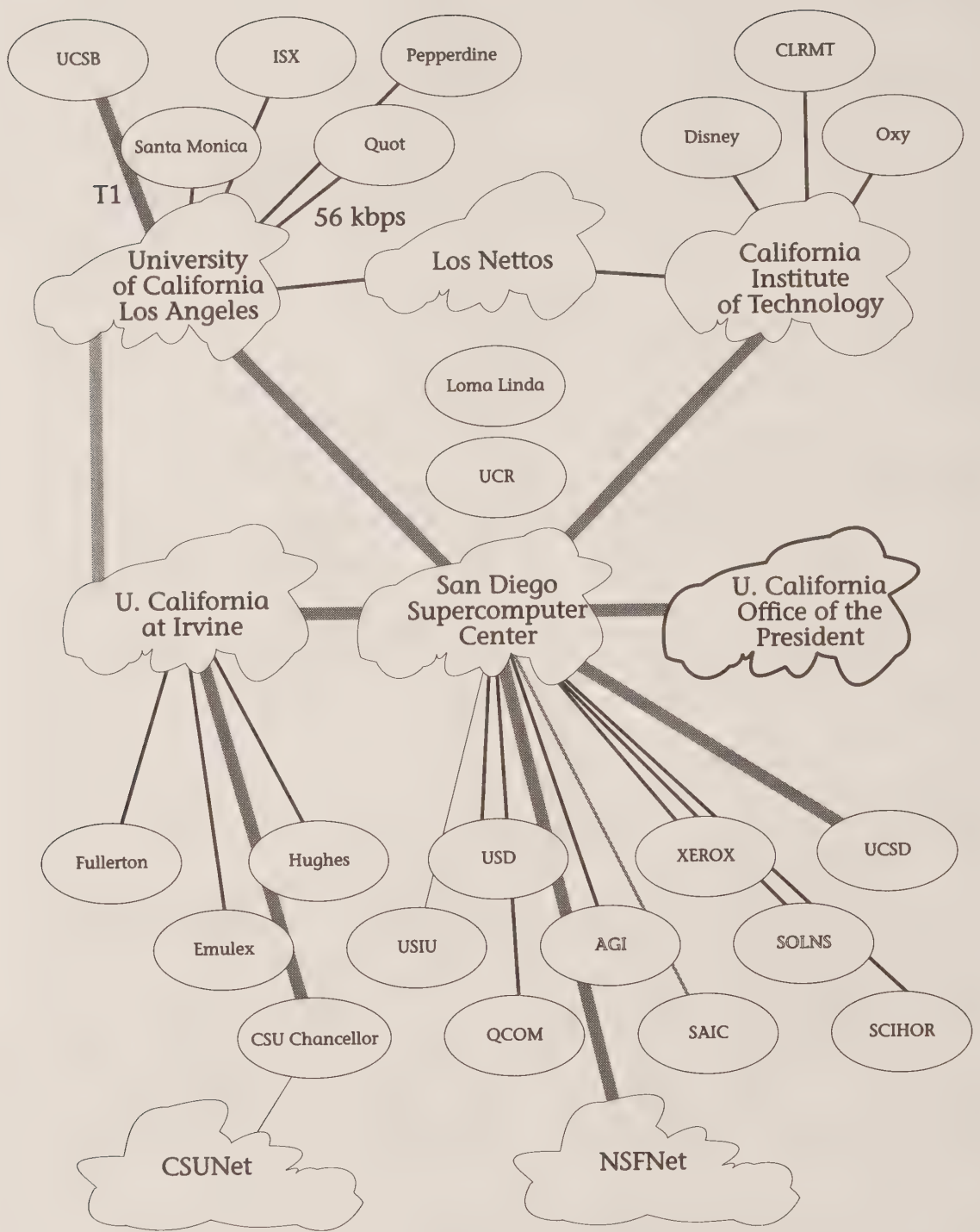
Most local users are on other Ethernets. For example, users who are doing visualization of results from the Cray computer may have their workstation on the visualization network. This network has a series of Sun and Silicon Graphics workstations, some frame buffers, and Alliant and VAX servers.

When access to more powerful resources is needed, traffic is routed over to a HYPERchannel system, running multiple links at 50 to 100 Mbps. The Hyperchannel is in turn connected to a CRAY YMP 8/864, which has 64 Gbytes of storage space.

Sixty-four Gbytes may seem like a lot for a workstation, but for a Cray computer it is insignificant. The NAS facility, for example, has 2 Gbytes of main memory plus a 2 Gbyte RAM disk. To feed the data appetite of such a

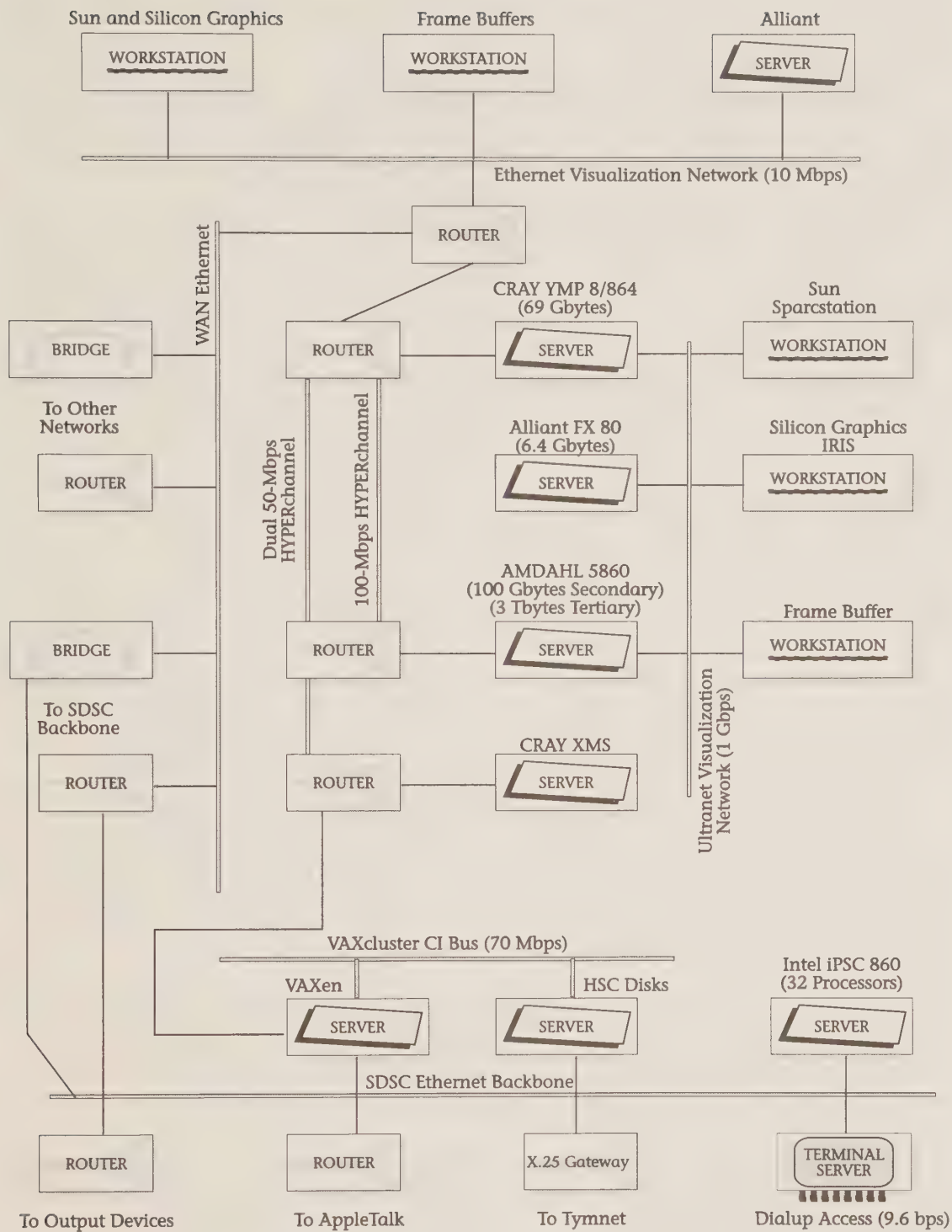


2-22 SDSC WAN Links



2-23 CERFnet





2-24 The SDSC Network

large system, an Amdahl mainframe has been dedicated as a storage controller.

The Amdahl has another 100 Gbytes of disk space. In addition, it is connected to an automated tape system that can give a user access to over 3 terabytes of data within 30 seconds. For the reader who would like to visualize this information, three terabytes is a stack of 3.5 inch floppy disks over 8 miles high.

The IBM file server and the Cray computer could communicate over the HYPERchannel, but can also use another path over an Ultranet network. This Ultranet network also has frame buffers, Sun workstations, and other systems. This high-speed visualization network also has an Alliant FX 80 to act as a file server.

A network like this is always evolving. Figure 2-25 shows the next phase for the SDSC network. Notice that the core networks are based on FDDI, one as the SDSC backbone, the other forming the backbone for the University of California at San Diego, where the SDSC is located.

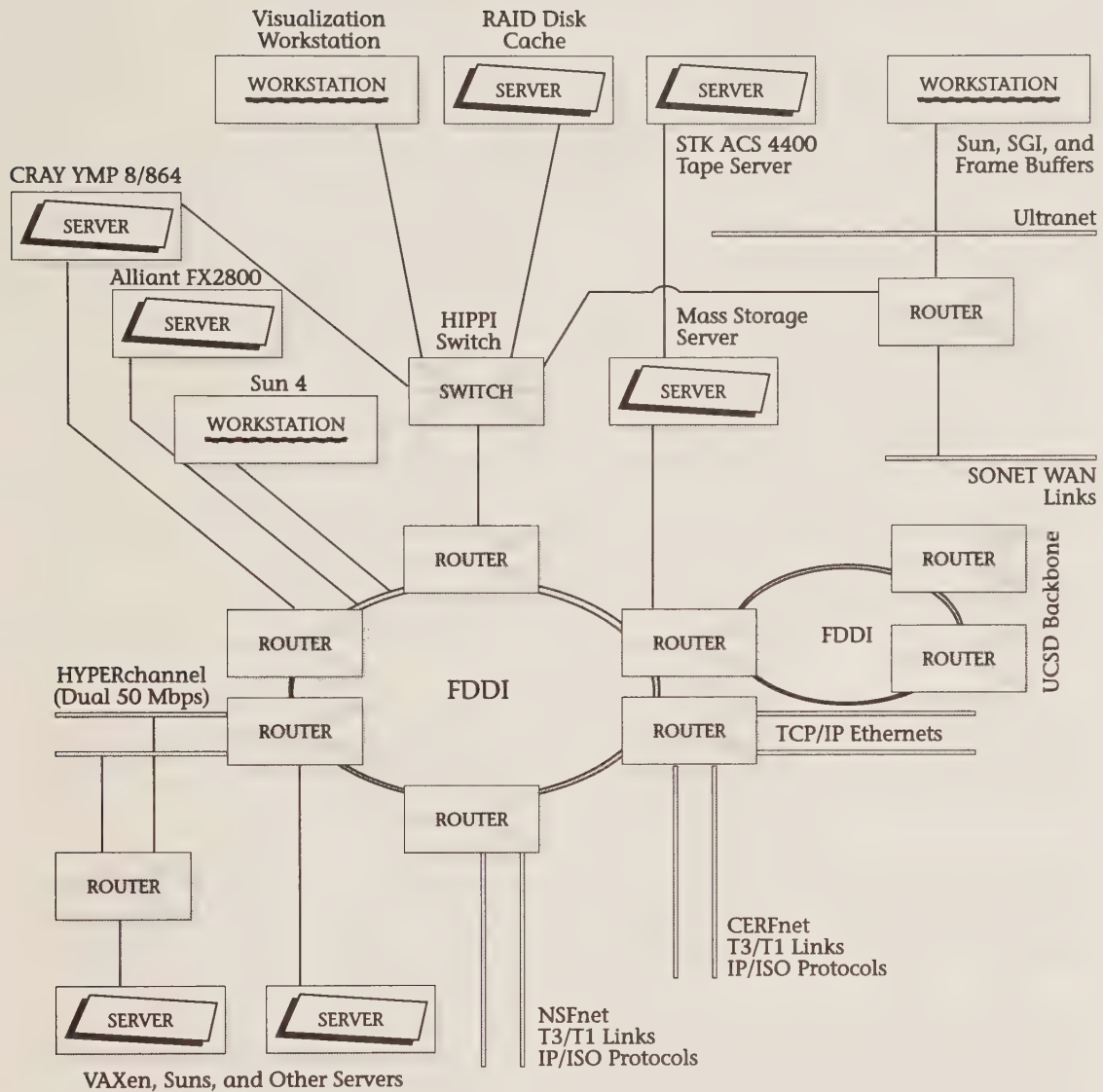
The FDDI backbone has links to CERFnet and NSFnet, routing both TCP/IP and OSI protocols. There are also a variety of WAN links to DECnet and other environments. The HYPERchannel is still there (lower left of the diagram) as is the Ultranet (upper right). In addition, however, there is a switch based on the High Performance Parallel Interface (HIPPI), which runs at 800 Mbps.

The switch, made by Network Systems Corporation, is actually a crossbar switch which physically sets up a circuit between two nodes. It has a total bandwidth of several gigabits. To that switch are connected the main server systems, as well as a router to the Ultranet and FDDI networks.

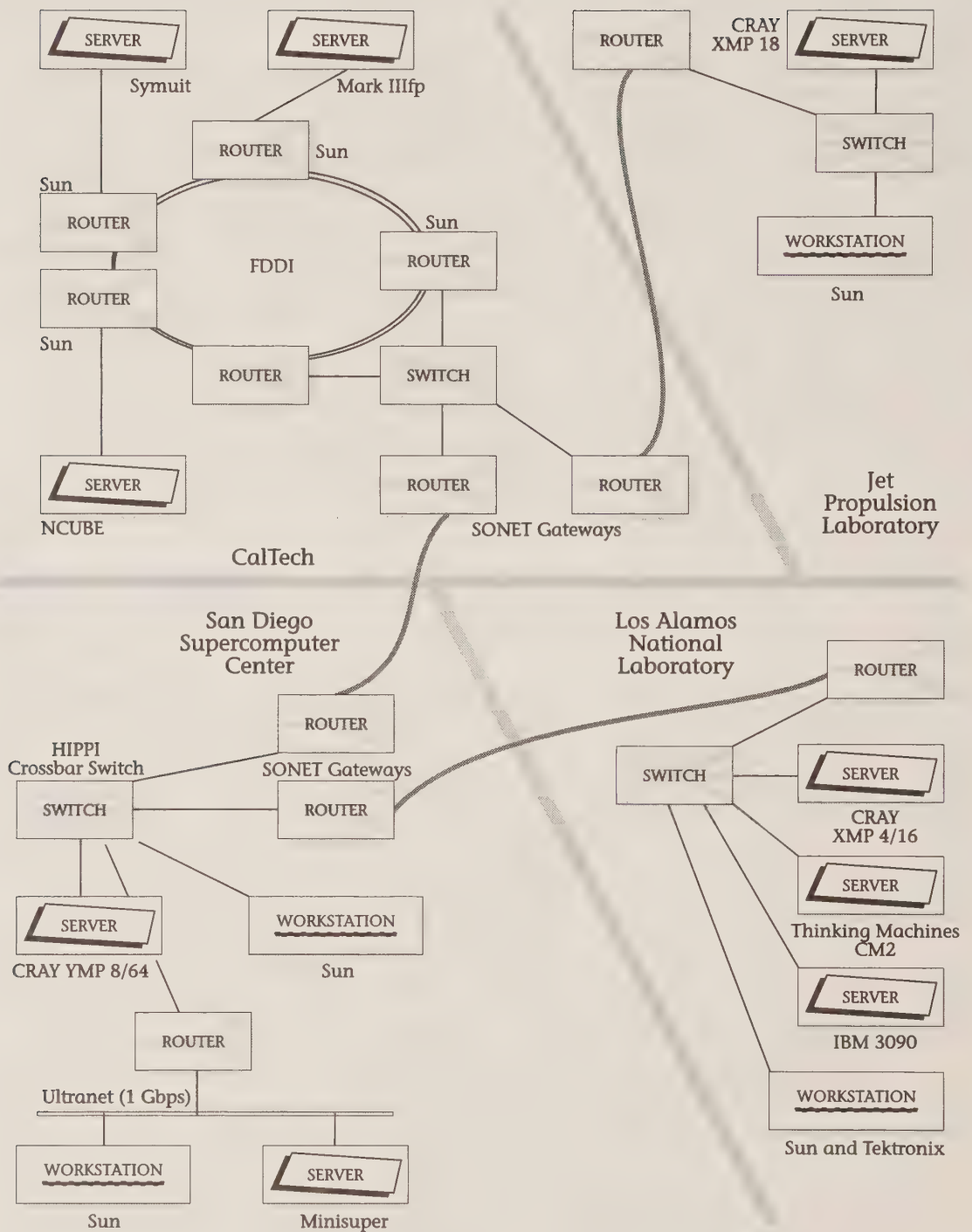
The HIPPI switch also has on it a disk cache based on a technique known as a Redundant Array of Intelligent Disks (RAID). The basic theory is that having a lot of cheap fast disk drives is better than having a single large drive. The small drives can operate in parallel, improving performance, and are cheap enough to allow redundant drives to protect data. The HIPPI switch can also be used to connect massively parallel processors or other supercomputers.

Figure 2-26 shows how the wide-area access for the network is also evolving. SDSC is a participant in a wide-area gigabit testbed called CASA. The other sites for this testbed are the California Institute of Technology (Caltech), the Jet Propulsion Laboratory, and the Los Alamos National Laboratory.

Links between the systems use SONET standards over fiber optic cable provided by a long-distance company. The SONET gateways to the WAN network would then connect to a facility's local HIPPI switch, which in turn connects to a network based on standards such as FDDI, HIPPI, or Ultranet.



2-25 Migration Path for SDSC Network



2-26 The CASA Gigabit Network



The CASA network helps solve a few pressing research problems that can't fit onto the facilities at any one of these supercomputer sites. In addition, CASA, along with four other testbeds, helps provide experience for a National Research and Education Network (NREN) operating at gigabit speeds.

## Packet Radio

We switch now from the Porsche to the Volkswagen for wide-area networks. Perhaps the most spectacular example of how easy it is to add new subnetworks into TCP/IP was demonstrated by Phil Karn in his KA9Q package. This public domain code added TCP/IP to the PC, which is widely used for dedicated routers or PC clients in networks.

The original intent, however, was to provide support for packet radio. Using radio waves to transmit data packets was originally part of the Aloha network in Hawaii.

Packet radio can also be part of the amateur (ham) radio network. Phil Karn estimates 30,000 people around the world now have equipment capable of capturing data. Ham radio is strictly amateur: nobody can charge or accept payment for the services they provide. Consequently, this is an environment that requires low-cost equipment. Ham radio uses a wide range of propagation techniques: direct line of sight, reflection from the ionosphere, orbiting satellite, or local hilltop reflector. It even uses the moon as a passive reflector.

A typical packet radio modem is part of a special-purpose box called a Terminal Node Controller (TNC). TNC also has a single-board computer with the firmware to execute the packet protocols. There are typically two ports: one is an RS-232 connection for host computer, and the other is a connection to the radio's audio I/O and push-to-talk leads.

Karn points out that packet radio channels are "radically different from the far more benign wire or fiber transmission path." Packet radio stations are very similar to Ethernet in that everybody can talk and there is a (somewhat nondeterministic) backoff method.

There are some strong differences, however. First, carrier sensing is not nearly as good as Ethernet. Not every station can hear the transmissions of every other station. In addition, a station that is transmitting cannot also receive at the same time (which makes collision detection somewhat difficult).

The biggest difference, however, is the quality of the signal. The bit error rate can be quite bad, and the signaling speed is much lower than Ethernet because of bandwidth and power limitations. To manage the signal, a special link level protocol called AX.25 is used. Note that this is not X.25, al-

though it has some elements of a subset of HDLC also used by X.25 called the Link Access Procedure B (LAPB).

AX.25 sends packets synchronously in HDLC frames. Typically, half-duplex operations are used since access to the channel is shared. The header has an address header which has at the very least the FCC-assigned call signs of both destination and source in ASCII. After the source field are up to eight “digipeater” addresses: these are intermediate stations that can receive, store, and retransmit a packet (usually on the same frequency). This is strict source routing: the sender specifies the entire digipeater string.

Next is the LAPB control field and the Protocol ID (PID) byte. PID is often F0, signifying that no upper-level protocol is in use and data should be sent directly to the terminal. Used this way, AX.25 is simply a way for two terminals to “chat.”

AX.25 uses two forms of encapsulation: Information (I) and Unnumbered Information (UI) frames. Both are needed: I frames are explicitly acknowledged and are used on fading, noisy 300-bps links for long-haul HF bands. UI frames would then be used for 56-kbps line-of-sight paths on UHF frequencies where the overhead of the I frames would be intolerable. Picking one mode over the other is through the use of the IP type-of-service (TOS) bits.

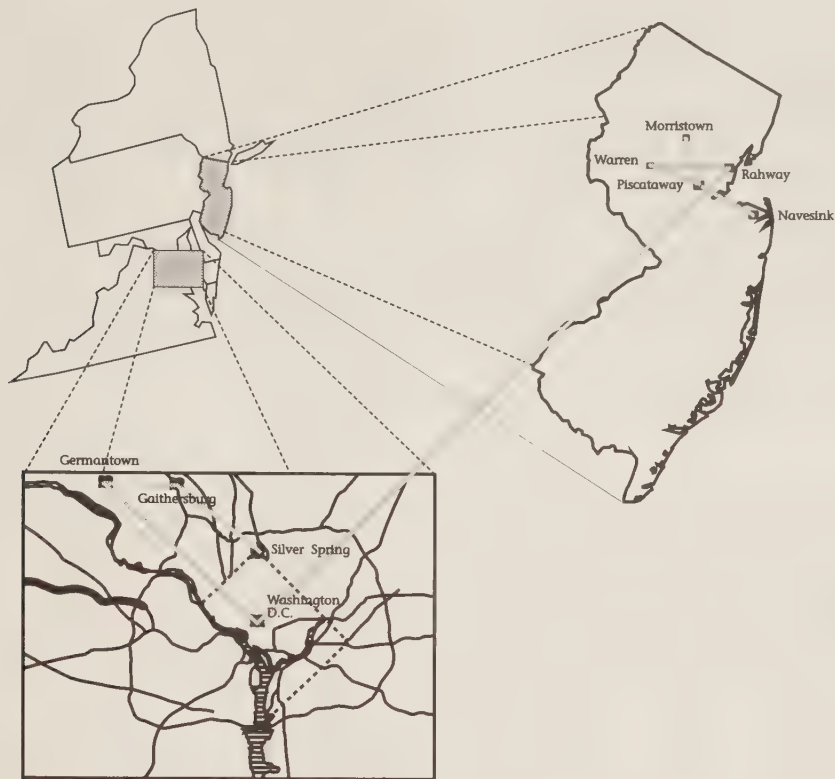
The first TCP/IP experiments were done by Richard Bisbey and Dave Mills (creator of the network time protocols). Both adapted existing IP gateway hardware and software.

Karn then created an Internet software package specifically for amateur packet radio use. The result is the KA9Q Internet Protocol Package. There is a registered domain (ampr.org), and a Class-A network address (network 44). All this is for the Amateur Packet Radio (AMPR) subnetwork/network.

In a typical network, the idea of subnetworks is very well defined: a series of data links all connected together into one network. The Amateur Packet Radio network (AMPRnet) consists instead of individual stations and ad-hoc links. Subnetworks are not interconnected as one would see in a more normal network. Instead, we have isolated islands.

The approach used to allocate the address space is “generalized subnetting”: the entire 32-bit address space is a subnetwork mask. Basically, every entry in the routing table has its own subnet mask. When you do a routing lookup, you return the matching table entry that has the widest subnetwork mask. This lets you construct an arbitrary network topology. A host-specific routing entry is a subnet mask of all 1s; a default route is a mask of all 0s.

This was all put together in the KA9Q Internet Protocol package. Since Suns and VAXen are somewhat rare in the ham world, the package was originally developed for the PC. The basic package supports IP, ICMP, TCP, UDP, and the big three services (Telnet, FTP, and SMTP). A variety of sub-

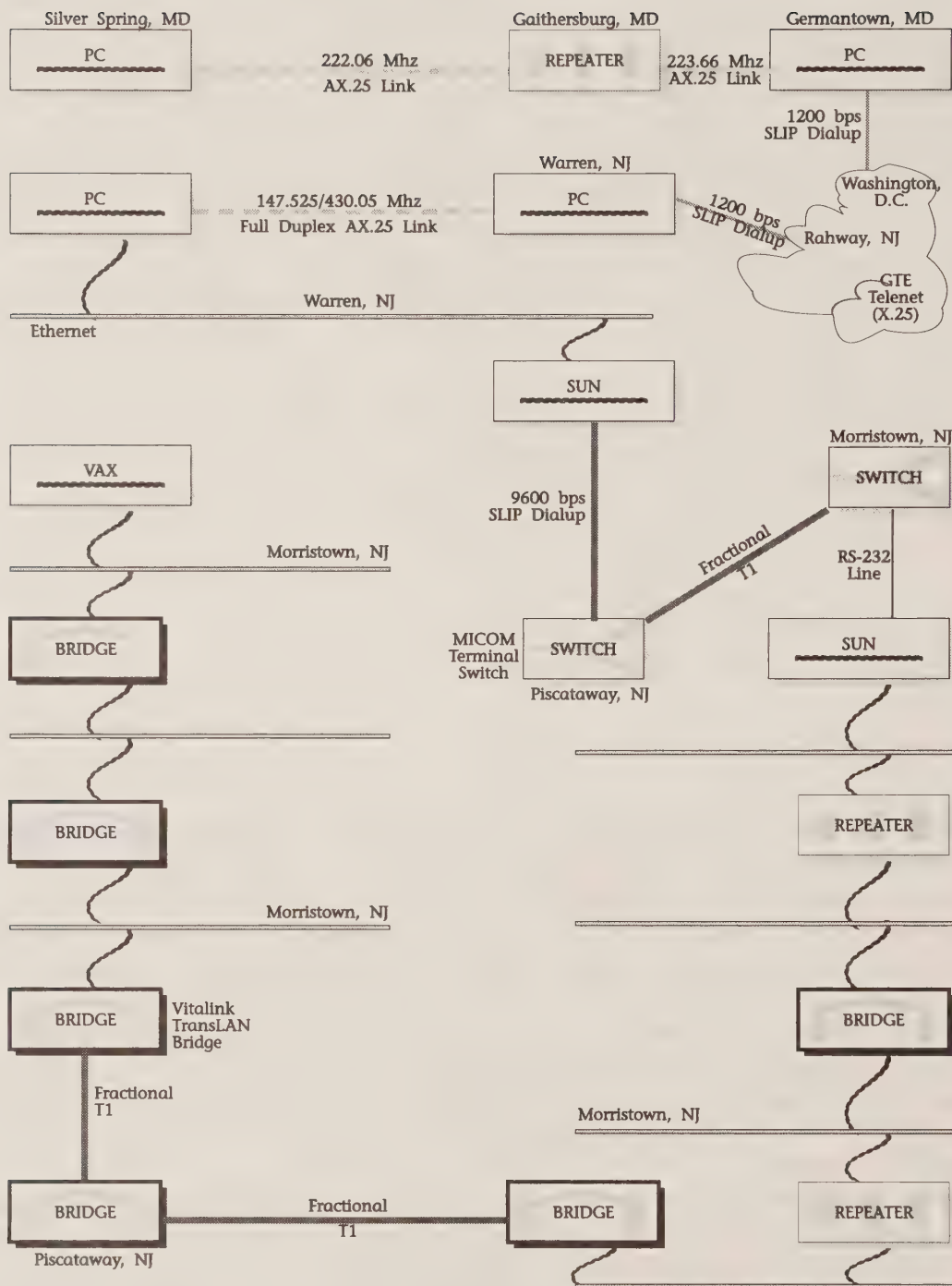


**2-27** The Amateur Packet Internet

network protocols in addition to AX.25 are supported including SLIP, PPP, and the 3Com Ethernet board. Although the original environment was MS-DOS, the code has also been ported to Apple Macintosh, Commodore Amiga, Atari ST, and various mutations of Unix System V. Since the original version, the package has been greatly enhanced through contributions from a wide number of users and much effort by Phil Karn.

Once the PC systems were up and running, a sample network was put together. The network used facilities in Washington, D.C., and suburbs, as well as in New Jersey (see Fig. 2-27). The link between the two areas was with an X.25 network (Tymnet), and a variety of links were used at either end (see Fig. 2-28).

This “amateur Internet” shows how simple it is to string together links. The network started in Silver Spring, Maryland, where an amateur radio link was set up to Germantown, Maryland (with a repeater in the middle). There, traffic was routed through an X.25 link to a PC in Warren, New Jersey.



2-28 The KA9Q Test Network



The New Jersey system then used packet radio to move packets through the Warren backbone over to a Sun-based router. There, the SLIP protocol was used to move data to a terminal switch, where the data finally arrived at another Sun router in Morristown, New Jersey.

Once in Morristown, packets moved through a series of repeaters, bridges, WAN-bridges, and more bridges, until they finally reached a VAX server. What did the user do at this point? That's the subject of the rest of the book. First, the TCP/IP protocols which integrate the data links into a coherent network will be examined. Then, the different services that can run over these links will be shown.

### For Further Reading

ANSI, "High-Performance Parallel Interface—Framing Protocol (HIPPI-FP)." X3T9.3/89-146, X3T9.3/89-013, Rev. 3.1, January 23, 1991.

———, "High-Performance Parallel Interface—Encapsulation of ISO 8802-2 (IEEE Std 802.2) Logical Link Control Protocol Data Units (HIPPI-LE)." X3T9.3/119-Rev. 2.0, December 3, 1990.

———, "High-Performance Parallel Interface—Physical Switch Control (HIPPI-SC)." X3T9.3/90-Rev. 1.6, February 28, 1991.

Jacobson, V., "Congestion Control and Avoidance," *Proceedings ACM SIGCOMM '88*, Stanford, Ca., August 1988.

Phil Karn, Amateur Packet Radio and TCP/IP, *ConneXions*, pp. 8-15, v. 2, no. 9, September, 1988.

Lang, Lawrence J., and Watson, James, "Connecting Remote FDDI Installations" *Computer Communication Review*, p. 72, v. 20, no. 3, July 1990.

Malamud, Carl, *STACKS*, Prentice Hall (Englewood Cliffs, NJ: 1991). Includes discussions of SMDS, HIPPI, and B-ISDN.

———, *Analyzing Novell Networks*. Van Nostrand Reinhold (New York: 1990). Includes discussions of ARCnet and Token Ring.

John Romkey, *A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP*, RFC 1055, June, 1988.

Ross, Floyd E., et. al., "FDDI: A LAN among MANs" *Computer Communication Review*, p. 16., v. 20, no. 3, July, 1990.

Mark Wolter, "Fiber Distributed Data Interface (FDDI) - A Tutorial," *ConneXions*, pp. 16-26, vol. 4, no. 10, October 1990.



## CHAPTER 3

# Networks





# Networks

## Networks

The previous chapter examined the network from the point of view of the data links that make up individual pieces: Ethernets, serial lines, X.25, FDDI, or any of the other different technologies available to connect computers together.

This chapter takes a step back and looks at the network again. A data link is meant to move a packet of data from one node on that data link to another one. Routers formed the gateway between two different data links. Some network applications can be confined to a single data link. Word processing, for example, might involve a workstation and a server, both located on a departmental Ethernet. In many cases, however, the two nodes that wish to communicate—the client and server—may be located on different subnetworks.

The network layer builds on the basic data link service to make any two nodes on the network accessible to each other. As with the data link service, we are forwarding a packet of data. What the packet means or who it is for is an upper-layer issue. The network layer is thus a connectionless network service, providing a best-effort delivery. This is the basic service provided by the Internet Protocol, the IP part of TCP/IP. A packet is sent to a router, which forwards it to another router, and eventually the packet is delivered to the destination node.

While the native interface for Sun computers is the IP protocol, it is also possible to run the OSI Connectionless Network Service (CLNS) network layer protocols by using SunLink gateway systems. The question of integrating multiple network (and transport) protocols will be returned to later in this chapter and explored in more depth in Chapter 16.

While the network layer gets data to the destination node (or at least tries), there is still a need for an even better service: communication between programs, not computers. A given program, an application, might be

in communication with a variety of different other applications, some local and some remote.

The transport layer is responsible for accepting incoming packets from the IP layer and sorting them out by users, each one identified by a port. There are a variety of different transport protocols. The User Datagram Protocol, used by applications such as NFS, provides just this basic service: data delivery to a program.

Other transport protocols, such as the Transmission Control Protocol (TCP), provide a higher level of service. TCP not only gets the data to its end destination, but it makes sure that the data arrives in order. Making sure it arrives in order is a guaranteed end-to-end delivery of data, a service that allows applications to assume that their messages were received.

In addition to the transport and network layer protocols, a variety of service protocols are also examined. Some are very basic in nature, such as the Address Resolution Protocol (ARP), which allows mapping of addresses from the network to the data link layer.

Other services are more sophisticated, such as routing protocols. Remember that we said a router accepts a packet and sends it on its way? In large networks, deciding which of the many possible data links and intermediate gateways to use can be a difficult decision.

A network, a combination of one or more data links, can be organized into routing domains. Within a routing domain, a protocol is used to inform all routers about reachable destinations and the best path to use. An example of such a protocol is RIP, the Routing Information Protocol.

In addition, there are interdomain routing protocols, such as the Exterior Gateway Protocol, that allow exchange of reachability information across routing domains. These “glue” protocols are important because the Internet is composed of many different autonomously administered networks. Exterior protocols allow an autonomous administration to decide exactly what connectivity it will give the outside world.

## The Internet Protocol

The Internet Protocol is a mechanism that allows a user (e.g., the TCP or UDP modules) to send a packet to the next gateway or the destination host. IP has two basic functions: addressing and fragmentation.

This is a connectionless network service, which means that each datagram is an independent entity. The Internet Protocol will forward a packet to the next place on the network, but will not guarantee service.

IP uses four types of mechanisms to help provide more sophisticated service:

- Type of service
- Time to live

Class A	First bit 0
	7-bit network number
	24-bit local address
Class B	First bits 10
	14-bit network number
	16-bits local address
Class C	First bits 110
	21-bit network number
	8-bit local address
Extended addressing mode	First bits 111

3-1

IP Address Modes

- Options
- Header checksum

The type of service is simply an abstract metric asking for a particular grade of service, based on reliability, throughput, or some indicator. Very few routers actually make their routing decision based on this metric; most simply ignore it and choose the same route for all types of packets.

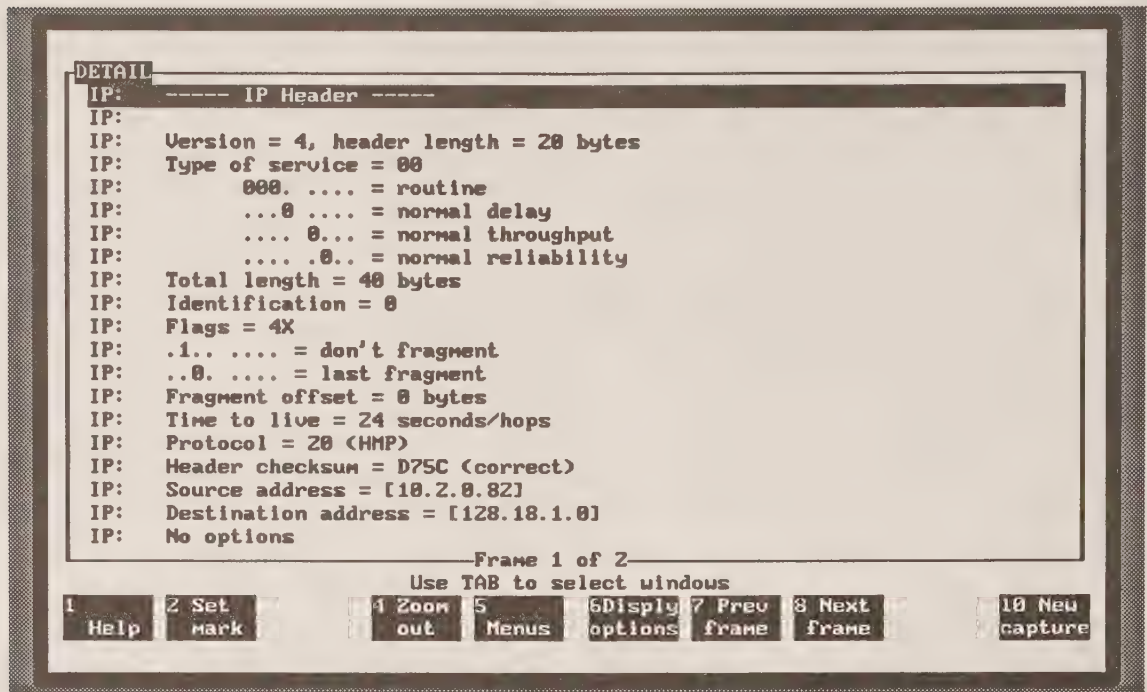
The time-to-live (TTL) indicator is an upper bound on the packet lifetime. Each point along the route reduces this number. When it reaches zero before reaching its destination, the datagram is destroyed. Options allow a per-packet service designation. Various options include things like time-stamps, security, or special routing considerations.

### *Addressing*

It is necessary to distinguish between names, addresses, and routes. A higher level, typically the naming service, gives the mapping from a logical name to an IP address. IP will map an internet address to a local net address. Routing is outside the scope of the IP module: determining how to get to some distant gateway is handled by the routing protocols.

The address is a fixed-length 32-bit quantity. It consists of a network number followed by local address (see Fig. 3-1). The IP protocol defines three types of addresses, each one allowing a different number of hosts in it. Class A is only for very large networks, whereas Class C is for networks of 255 hosts or less. The Class B network space is the most desirable for many environments, and it appears that the addressing authorities may run out of Class B addresses. When this will happen depends how fast the Internet grows.





3-2 An IP Header

Some Internet hosts are gateways and thus have several distinct internet addresses, one for each network they are on. Note also that some hosts have several physical interfaces. Therefore, be careful when mapping logical IP addresses to local net addresses. You must provide for several physical interfaces with each having several logical internet addresses.

### Basic Operation

Figure 3-2 shows an example of the IP header for a packet. This IP packet would be handed down to a data link layer for delivery to another IP node, the destination host, or an intermediate router. The IP layer, in turn, could be carrying data for an upper-level user such as the TCP transport protocol.

If the destination host is on the same data link, the IP packet is handed to the data link layer with instructions to deliver it directly to the destination. In this case, the address on the data link layer will correspond to the IP address in the packet.

If the destination host is on another network, however, the packet is sent via the data link service to some intermediate router. The router will see that the destination address for this IP packet is some other node and will make the appropriate forwarding decision to move the packet one hop closer to its final destination.



Notice that there is a time-to-live (TTL) field inside the IP packet header. Every host that receives this packet decrements the TTL field by at least 1. When the TTL field reaches zero, it is summarily thrown away. Remember, the IP service is not a guaranteed service—delivery of data is guaranteed at the transport layer, not at the network layer in this case.

The packet also contains a protocol address which corresponds to the upper-level user of the IP service. IP provides service to several different modules, including the TCP and UDP transport layers and the various support protocols such as the Internet Control Message Protocol (ICMP).

The last portion of the IP header is reserved for options (see Fig. 3-3). Options are used to control exactly how this particular packet is sent over the network. Notice, for example, the strict source routing option. This option contains an exact list of intermediate routers that will be used to carry this packet. If one of those systems is unavailable, the packet is thrown away.

### *Fragmentation*

It is possible to mark packets with a don't fragment flag. If that packet reaches a gateway with a link that has an MTU that is too low, it simply discards the packet rather than fragment it.

The ID field in the IP header is used to identify the logical segment, which consists of multiple fragments. The fragment offset and length show the position of a particular fragment within the overall segment.

Once a packet is fragmented in route, it is not reassembled until it gets to its final destination. Since each fragment might go by a different route during times of unstable topologies or if a router does load balancing, it doesn't make sense to worry about reassembly until the final destination is reached.

### *Subnetworks*

The concept of “the network” quickly fell by the wayside as administrative entities found they had to delegate control to other groups over portions of the networks.

As far as the outside world is concerned, the entire network is in fact a single network, serviced by one or several gateways. However, once data gets inside of the network, it must get routed, using gateways, to the proper destination subnetwork.

A network is thus a collection of subnetworks. A gateway connects subnetworks together, just as gateways connect networks together. As far as the routing is concerned, there is really no difference between a subnetwork and a network; the decision still has to be made which gateway is the next best hop for a particular destination.

IP Option Fields			
option-type octet			3 fields
	1 bit		copied flag (0 = not copied)
	2 bits		option class
			0: control
			1: reserved
			2: debugging
			3: reserved
	5 bits		option number
Option Class	Number	Length	Description
0	0	-	End of option list: 1 octet total (no length)
0	1	-	No operation.
0	2	11	Security
0	3	var.	Loose Source Routing
0	9	var.	Strict Source Routing
0	7	var	Record Route
0	8	4	Stream ID
2	4	var	Internet Timestamp
Security Field			This option MUST be copied on fragmentation. There may be only one copy of this option in a datagram.
Security Field	16 bits	Specifies 1 of 16 levels of security (of which 8 are reserved for future use. All 0s is unclassified. Others are confidential, restricted, secret, and top secret.	
Compartments	16 bits	Security compartments.	
Handling Restrictions	16 bits		
Transmission Control Code	24 bits	Used to segregate traffic and define controlled communities of interest.	
Loose Source and Record Route Option (LSRR)			
Option Code		type = 131	
Length Octet			
Pointer Octet		Route data is a series of addresses. If pointer > length, source route is empty (or recorded route is full) and routing should be based on destination. Every time field is incremented, source route is replaced with the recorded route. Loose route means that a gateway can use other gateways to reach the next address in the route.	
Route Data		List of addresses.	

### 3-3 IP Option Fields

<b>Strict Source and Record Route</b>		Can't use intermediate gateways not specified in list.
Option Code		type = 137
<b>Record Route</b>		Originating host must compose this option with a large enough data block to hold the route information. If the area is full, then you keep on forwarding without inserting the data.
Option Code		type = 7
<b>Timestamp</b>		
Option Code		type = 68
Option Length		Option length has maximum of 40.
Pointer		Pointer to end of overwritten area. Smallest legal value is 5.
Overflow Flag	4 bits	Nnumber of IP modules that cannot register timestamps due to lack of space.
Flag	4 bits	0: timestamps only
		1: timestamp preceded by internet address
		3: internet address fields are prespecified and you only put in your timestamp if your address matches the next one in the list.
Timestamp	32 bits	Right justified in milliseconds since midnight UT. If you have a different timestamp, you put in what you have but put the high order bit to 1 to indicate a nonstandard value.

### 3-3 (Cont.) IP Option Fields

To handle the question of subnetting, a few bits are stolen from the host part of an address. A Class B address, for example, is normally a 14-bit network number (assigned to the organization), plus 16 bits of host address space.

Instead of having 16 bits of address space, a network might designate 8 bits as the subnetwork address and the remaining 8 bits as the host space. This allows up to 255 different subnetworks, each having up to 255 different hosts. By convention, a zero means "this" entity: this network, this subnetwork, or this host.

An organization can apply for a Class B network address. Then, using the subnetting procedure, they can create the equivalent of 255 Class C networks out of that space. To the outside world, this is the equivalent of a single network. Once inside this administrative domain, there is further routing to get to the appropriate subnetwork.



When a host has to decide where to send a packet, the basic question is whether the packet is local or not. If the packet is going to a host on the same subnetwork, the host can send the packet itself: it sends an ARP packet to resolve the IP address into a data link address and then sends the packet.

If the packet is not local, it goes to a gateway, which will then send the packet on its way. The host decides if a particular address is local or remote by using a 32-bit mask, the host's own IP address, and the destination IP address. The subnetwork mask is a variable-length mask, allowing variable-length subnetworks.

The basic procedure is to take the fields on the subnetwork mask that are set to 1 and treat those digits as significant. If the first 8 bits are set and the 8 significant bits are the same, the destination address is on the same subnetwork as the local host.

How does a host find out its subnetwork mask? There are several ways to do this. One is the ICMP Address Mask Request and Address Mask Reply. The Boot Parameters (BOOTP) protocol also includes a provision for the subnetwork mask.

## ISO CLNS and TCP's IP

The two major network layer protocols are the IP protocol and the Connectionless Network Protocol (CLNP) used in OSI environments. Both are supported on Sun workstations. IP is the native protocol, but CLNP is added with the SunLink products.

Rob Hagens of the University of Wisconsin published a comparison of the two protocols and showed that the two have roughly the same functionality at the service interface. Although the user sees no difference, there certainly are some internal differences.

One of the most obvious differences is in the size of the address space. IP uses a fixed length address of 32 bits, whereas ISO uses a variable-length address of up to 20 bytes (160 bits). This is a bit of a difference in size.

The Internet is currently rationing Class B addresses, those used for moderately-sized networks. In other words, the most desirable portions of the 32-bit address space are filling up. In the ISO world, it is unlikely that the 20-byte address space will be quickly used up unless very wasteful allocations are made. On the other hand, 20-bytes is a lot of extra baggage to carry around for a 1-byte piece of Telnet data.

Other addressing differences lie in both the address format and the location of the address. In IP, the maximum header size is 60 bytes; in ISO it is 255 bytes. This is because ISO makes extensive use of different kinds of options. The use of options can be good and bad. For example, fragmentation information in CLNP is an option. This can be taken out of unfrag-



mented packets, saving 6 bytes. In CLNP, all options are copied into fragments. In IP, each option has a flag indicating if it needs to be copied into a fragmented packet. Both protocols support similar types of parameters. For example, both have security parameters.

Both protocols also support strict and loose source routing. In IP, you actually replace the destination address field (which then is replaced again when you consult the source routing list). In CLNP, you leave the destination address alone (but do put in the intermediate address).

Both have a record route option. In IP, the addresses are listed in traversed order and can be reversed to find the return path. In CLNP, the route is recorded in reverse order. In IP, an implicit record route is obtained whenever a source route is used: when a source route address is removed from the list, it is replaced by the address of the host just visited (up to the length available in the field, then you stop).

Quality of Service (QOS) is treated differently. In IP, the type of service (TOS) is part of the header and includes flags for precedence, delay, throughput, and reliability. In CLNP, QOS parameters are included in the options. The three types of QOS indicators are source address-specific information (semantics undefined), destination address-specific information (semantics undefined), and globally unique information. CLNP thus has more flexibility in defining QOS, but it is harder to locate and use.

CLNP has globally unique information, such as the congestion experienced bit. There is no equivalent in IP, but there is an ICMP source quench message. Note that IP has a precedence parameter in the IP TOS. In CLNP, this goes in a distinct Priority Option.

For errors, IP uses ICMP messages whereas CLNP has an internal error report function, which uses a different kind of packet. Both mechanisms allow reporting of the following errors:

- Destination unreachable
- TTL reached zero
- Syntax error/parameter problem

CLNP doesn't have direct provision for things like redirect. All these functions are handled in the ISO world by a separate protocol, ES-IS (ISO Information Standard 9542). ICMP Source Quench, Echo, Timestamp, and Information Request have no direct counterpart at the network layer of ISO.

### Support Protocols: ARP and ICMP

Each node on our network has at least two addresses: one a data link address such as the 48-bit Ethernet address; the other an internet address, composed of a 32-bit network and host number.

The IP layer makes routing decisions based on IP addresses. A routing table indicates the IP address of some intermediate router, for example, that is used as a default routing decision. A packet for some unknown host is sent to this router.

The problem with this is that the default router is known only by an internet address. To send a packet to this host, we need to give the data link layer the data link address we want. The Address Resolution Protocol (ARP) solves this bootstrap problem.

ARP exists for all the multiaccess data link layers that IP supports, such as SMDS, FDDI, and Ethernet. Figures 3-4 and 3-5 show examples of an ARP request and an ARP reply. In this case, the sender knows its own addresses and includes the information in the request. This allows the destination node to cache the information, which will be needed for any IP replies.

In an ARP packet, we know the internet address of a destination and we are trying to find the hardware address for that data link. A similar problem is the Reverse Address Resolution Protocol (RARP), where a host knows its hardware address but not the IP address. This situation occurs when a diskless node boots itself up. It knows the hardware address because this information is configured in the LAN adapter ROM. The IP address is needed so that IP and TCP can be set up, which will then allow the operating system to be transferred over so the system can boot itself up.

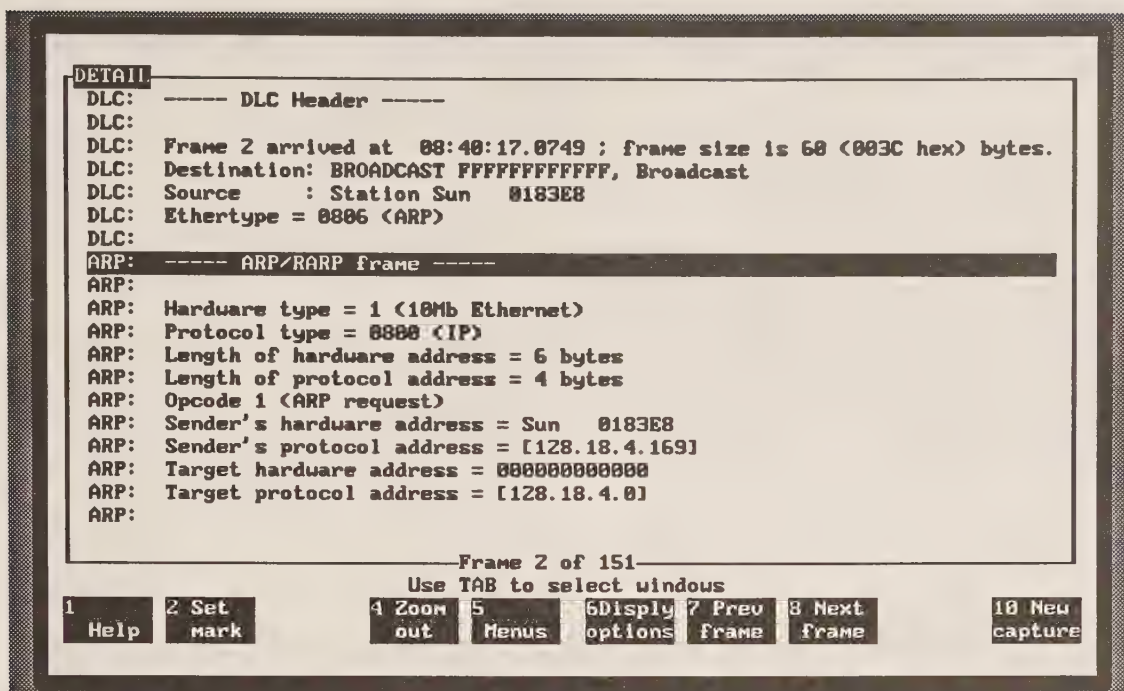
## ICMP

The Internet Control Message Protocol (ICMP) is a service built on top of the Internet Protocol. Although it is a separate module, it is an integral part of IP and must be included in every implementation. Of course, there are a few very primitive operations that don't support this—diskless booting using the BOOTP and Trivial File Transfer Protocol (TFTP) services being the notable examples.

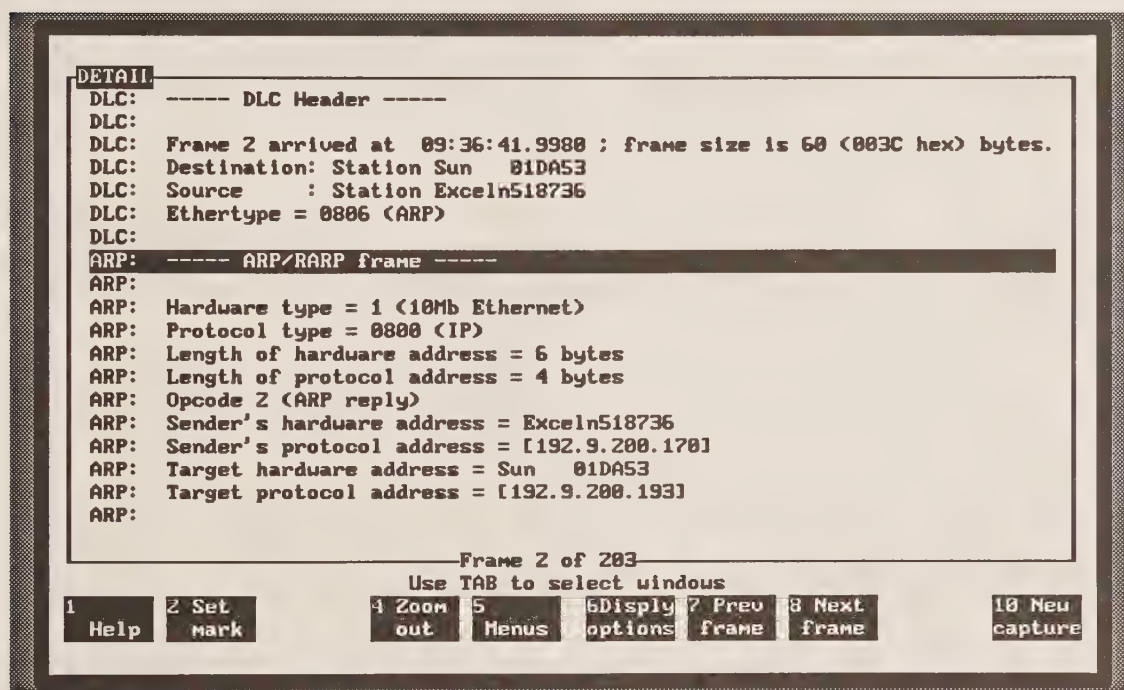
Remember that the IP protocol has no guarantee of service; packets are sent on a best-effort basis. ICMP provides a bit of feedback to the sending hosts, but, since it uses IP as the delivery mechanism, this feedback is of course on a best-effort basis.

ICMP messages are sent when an error occurs or other advice needs to be given to the sending host (see Fig. 3-6). No ICMP messages are ever sent about ICMP messages to avoid loops. ICMP messages are only sent for errors involving fragment 0 of fragmented datagrams.

ICMP includes three types of messages. A destination unreachable is sent back when a router is unable to move a packet on through to its destination (or at least one hop closer to its destination). This message is sent back to the source address of the offending datagram (see Fig. 3-7).



3-4 ARP/RARP Frame



3-5 ARP Reply



Destination Unreachable Message		
Type	1 byte	type=3
Code	1 byte	0: net unreachable
		1: host unreachable
		2: protocol unreachable
		3: port unreachable
		4: fragmentation needed but DF set
Checksum	2 bytes	5: source route failed
Unused	32 bits	
Original Data		Internet header plus 64 bits of original data
Time Exceeded Message		
Type		type = 11
Code		0: time to live exceeded in transit
		1: fragment reassembly time exceeded
Checksum		
Original Data		Internet header plus 64 bits of original data
Parameter Problem Message		
Type		type=12
Code		0 = pointer indicates the error
Checksum		
Pointer		If code is 0
Original Data		Internet header plus 64 bits of original data.
Source Quench Message		
Type		type = 4
Code		code = 0
Checksum		
Original Data		Internet header plus 64 bits of original data
Redirect Message		
Type		type = 5
Code		0: redirect datagrams for the network
		1: for the host
		2: for the type of service and network
		3: for the type of service and host
Checksum		
Address	32 bits	Gateway internet address
Original Data		Internet header plus 64 bits of original data

### 3-6 ICMP Messages



Echo or Echo Reply message		
Type		8 for echo
Code		code = 0
Checksum		
Identifier	2 bytes	Used to match echo and reply, can be 0.
Sequence Number		More matching information. Typical ID is the TCP/UDP port, the sequence number might be incremented on each echo request sent.
Data		Used to echo back.
Timestamp or Timestamp Reply Message		
Type		13 for timestamp, 14 for reply.
Code		code=0
Checksum		
Identifier	2 bytes	
Sequence number	2 bytes	
Originate Timestamp	32 bits	Last time sender touched the message.
Receive Timestamp	32 bits	Time echoer first touched it on receipt.
Transmit Timestamp	32 bits	Time echoer last touched it to send it.
Information Request or Reply Message		
Type		15 for info, 16 for reply.
Code		code = 0
Checksum		
Identifier		
Sequence Number		

### 3-6 (Cont.) ICMP Messages

The time-exceeded and parameter problem messages are used when IP might know how to route the packet, but can't. If the time-to-live (TTL) field reaches zero, the time-exceeded message is sent. If there is some formatting problem, then the parameter problem report is sent.

Two other messages, the source quench and redirect messages, provide feedback to the host on the current state of the network. If a router is about to reach capacity, the source quench message is sent back in the hope that the sender will back off. There is no guarantee that the sender will back off, but many TCP implementations are able to adjust their transmission rates in response to this message.

The redirect message occurs when a host sends a message to a gateway for processing (see Fig. 3-8). The gateway looks in its routing tables and sees that another gateway can do a better job and that the second gateway is directly accessible (on the same subnetwork) as the sending host. Sending a redirect message informs the sending host that it should use the second gateway in the future when trying to reach the destination.

The original packet is forwarded directly to the second gateway. Remember that ICMP is advisory. If the sending host wants to ignore the redirect, it will just keep sending to the first gateway. Such a poorly formed implementation would probably work although it would violate a number of provisions in the standards.

A few other miscellaneous message types are also available. The echo and echo reply, for example, are used by the “ping” command to test to see if a destination is available (see Fig. 3-9). The timestamp messages are a way of finding a host’s current clock reading, although we will look at the Network Time Protocol as a much more sophisticated way to handle this problem.

## TCP

The Transmission Control Protocol (TCP) is a connection-oriented, end-to-end reliable protocol layered on top of IP. IP gets the data to the final destination. TCP then takes over and makes sure all the data made it and in the right order.

The basic service to the user of TCP—an application—is a continuous stream of octets in each direction. The user doesn’t worry about how the data gets to the destination (IP’s concern) or even about things like the maximum transfer unit (MTU).

The basic TCP service takes incoming packets from the IP module. There may be several different users of TCP: TCP will thus take the incoming packets and allocate them to each connection using a particular pairing of users on two nodes. The access point to TCP is known as a port.

Once TCP has allocated packets to the relevant port, it checks a sequence number to see that the data stays in the proper order. In addition, after receiving data TCP generates an acknowledgment so the remote node will know that its data has made it.

In addition to reassembling the incoming data, the TCP module also keeps track of outgoing information. Acknowledgments are correlated against the outgoing data. Once a piece of outgoing data has been acknowledged, the TCP module can throw that data out.

If for some reason a piece of data is not acknowledged, however, the sending TCP module will retransmit all data that has not been acknow-

## DETAIL

ICMP: ----- ICMP header -----

ICMP:

ICMP: Type = 3 (Destination unreachable)

ICMP: Code = 0 (Net unreachable)

ICMP: Checksum = 3FCA (correct)

ICMP: IP header of originating message (description follows)

ICMP:

IP: ----- IP Header -----

IP:

IP: Version = 4, header length = 20 bytes

IP: Type of service = 00

IP: 000. .... = routine

IP: ...0 .... = normal delay

IP: .... 0... = normal throughput

IP: .... .0.. = normal reliability

IP: Total length = 85 bytes

IP: Identification = 3126

IP: Flags = 0X

IP: .0.. .... = may fragment

IP: ..0. .... = last fragment

Frame 23 of 26

Use TAB to select windows

1 Help

2 Set mark

4 Zoom out

5 Menus

6 Display options

7 Prev frame

8 Next frame

10 New capture

## 3-7 ICMP Destination Unreachable

## SUMMARY

Delta T DST SRC

10	0.0845	Intrln042DF4+Sun	00394E	SNMP Get ifOperStatus ... ifOper
11	0.0289	Sun	00394E+Intrln042DF4	SNMP Got ifOperStatus ... ifOper
12	0.1160	Intrln042DF4+800010033B27		SNMP Got tcpConnRemAddress ... t
13	0.0023	800010033B27+Intrln042DF4		ICMP Redirect (Redirect datagram)
14	0.0008	Sun	00394E+Intrln042DF4	SNMP Got tcpConnRemAddress ... t
15	0.0067	800010033B27+Sun	00394E	SNMP Next tcpConnRemAddress ...
16	0.0631	PrteonE04033+PrteonE04033		Ethertype=7030 (Interlan tes
17	0.1647	Intrln042DF4+800010033B27		SNMP Got tcpConnRemAddress ... t
18	0.0023	800010033B27+Intrln042DF4		ICMP Redirect (Redirect datagram)

## DETAIL

ICMP: ----- ICMP header -----

ICMP:

ICMP: Type = 5 (Redirect)

ICMP: Code = 1 (Redirect datagrams for the host)

ICMP: Checksum = D5D9 (correct)

ICMP: Gateway internet address = [130.128.1.35]

ICMP: IP header of originating message (description follows)

ICMP:

IP: ----- IP Header -----

Frame 13 of 964

Use TAB to select windows

1 Help

2 Set mark

4 Zoom in

5 Menus

6 Display options

7 Prev frame

8 Next frame

10 New capture

## 3-8 ICMP Redirect Message



The screenshot displays a network analysis interface with a 'SUMMARY' table and a 'DETAIL' section.

SUMMARY	Delta T	DST	SRC	
16	2.9282	Broadcast	+DEC 03CF8D	RWHO HOST=hania
17	3.9945	Broadcast	+DEC 033482	RWHO HOST=astrovax
18	1.0338	Broadcast	+Intrln003133	RWHO HOST=acm-cel
19	4.4888	DEC 024B1C	+Intrln0073AB	ICMP Echo
20	0.7565	Broadcast	+Sun 017525	RIP R Routing entries=2
21	5.2692	Broadcast	+DEC 035189	RWHO HOST=merci
22	1.9212	Broadcast	+DEC 0298C4	RWHO HOST=puvax1
23	1.4654	Broadcast	+Sun 013171	RWHO HOST=mira
24	1.7904	Broadcast	+U-B 39B900	RWHO HOST=put2

**DETAIL**

```

ICMP: ----- ICMP header -----
ICMP:
ICMP: Type = 8 (Echo)
ICMP: Code = 0
ICMP: Checksum = 02CF (correct)
ICMP: Identifier = 63526
ICMP: Sequence number = 25603
ICMP: [56 bytes of data]
ICMP:
  
```

Frame 19 of 48  
Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

### 3-9 ICMP Echo

ledged. It is also possible for the remote site to make a negative acknowledgment, again forcing retransmission.

The amount of data that remains unacknowledged is known as a window. The window can be different for each direction. The receiver of the data will offer a window, which is authorization for the sender to send a certain amount of data.

The sender also keeps track of a window, but it takes the offered window from the receiver and subtracts all data that has been sent but not acknowledged. This is known as the usable window.

When a connection is established, a bit in the TCP header (see Fig. 3-10) is set to synchronize (SYN) the connection. Along with this flag, the node sends its initial sequence number (ISN). The ISN is the start of a data stream. Every byte that is sent in this connection is some offset from the ISN. If the initial sequence number is 1000 and 2000 bytes are sent, the next sequence number is 3001.

Each side of the full-duplex connection gets an ISN, each side being both a sender and a receiver. When data is sent, the sequence number is included, allowing the receiver to reassemble the data stream and detect missing pieces.

When a TCP module sends data, it can also include an acknowledgment (ACK), which is the highest sequence number that this module has received.



Source Port	2 bytes	
Destination Port	2 bytes	
Sequence Number	4 bytes	Unless SYN is present in the flags field, this is the number of the first data octet in the data field. If SYN (synchronize sequence numbers) is set, this is the INITIAL SEQUENCE NUMBER (ISN) and the first data octet would be ISN+1).
ACK Number	4 bytes	If the ACK bit is set (ACK field significant) this is the value of the next byte expected. This is always set once the connection is established.
Flags	64 bits total	
Data Offset	4 bits	Number of 32-bit words in header.
Reserved	6 bits	
Control Bits	6 bits	URG: urgent pointer field significant
		ACK: ACK field significant
		PSH: Push
		RST: Reset the connection
		SYN: synch sequence numbers
		FIN: no more data from sender
Window	16 bits	Number of octets past the one ACKed this sender is willing to accept.
Checksum	16 bits	
Urgent Pointer	16 bits	Value is an offset within this segment: points to the octet FOLLOWING the urgent data.
Options	variable-length	
Padding	variable-length	Makes header a multiple of 32 bits.
Data		

### 3-10 TCP Header Format

Piggybacking an ACK on an outgoing packet saves bandwidth. If necessary, an ACK can travel in a packet with no data, but of course the overhead is higher since the ACK needs a full TCP (and IP and Ethernet) header.

Management of these sequence numbers requires keeping track of several pieces of information. A sending node needs to keep track of four numbers:

- Old sequence numbers that have been ACKed
- Old sequence numbers not yet ACKed
- Sequence number to use for next data transmission
- Future sequence numbers that can't be used yet

The last number solves the wrap-around problem. The ISN is picked so as to be fairly random. At some point, especially on long connections, the sequence number will reach all 1s and the number will wrap-around.

If the connection is going very fast, wrap-around may occur very quickly. In fact, on very fast, very long data links (or any path that involves one of these data links) it is even possible for the outgoing sequence number to catch up to some unacknowledged data.

The reason for selecting an initial sequence number at random is that there are no restrictions on reusing a connection over and over. A connection is simply a pair of sockets or stream heads: the two different kinds of entry points to the network.

Many sites tie the initial sequence number to the system clock. If a 32-bit sequence space increases every 4 microseconds, the 32-bit sequence will wrap around in 4.55 hours, usually long enough to prevent a packet from living that long.

Connection establishment is fairly simple in TCP. The connection establishment phase costs three TCP packets. The SYN bit is sent by the initiating node and a sequence number is sent. The other node acknowledges the sequence number and sends one of its own. The first node then ACKs the other node's sequence number. Following this initial handshake, both nodes are in sync and they can proceed.

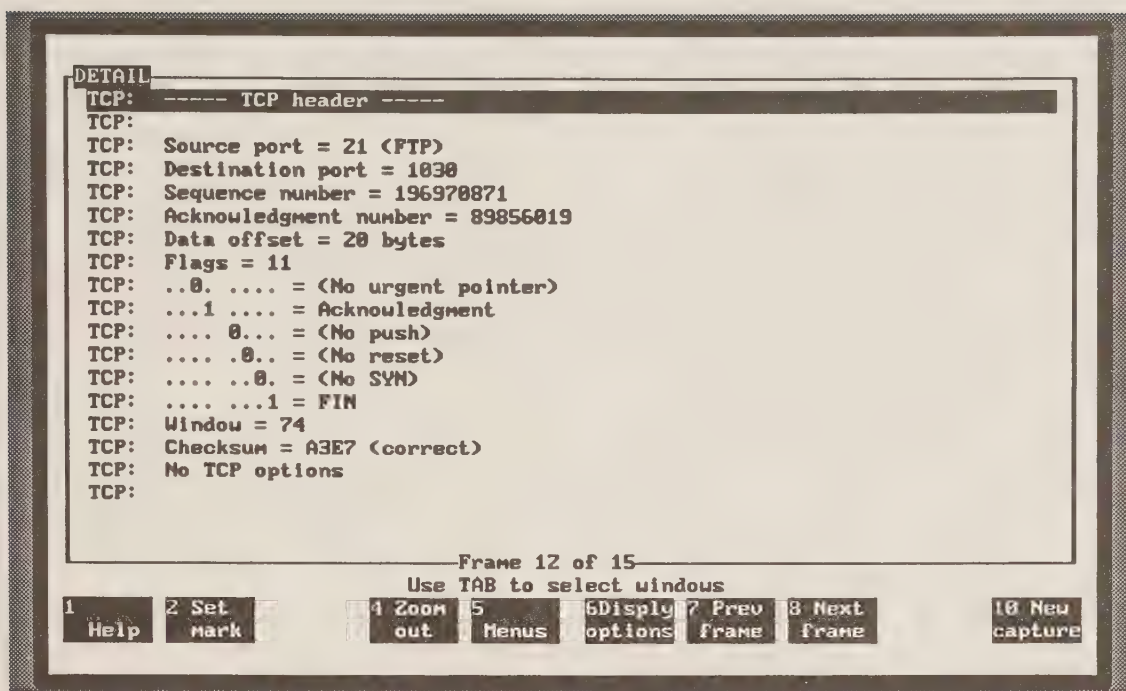
Figure 3-11 shows a typical TCP header embedded within an IP packet, which would in turn be embedded inside a data link packet (Ethernet, in this case). Notice that the source port is FTP, the File Transfer Protocol. The destination for this packet is an unknown port, most likely the FTP client.

The flags indicate that this packet is a simple acknowledgment packet and carries no user data. In addition to furnishing an acknowledgment, this packet has a sequence number. The sequence number will be used by the remote node so that it can send its own acknowledgment.

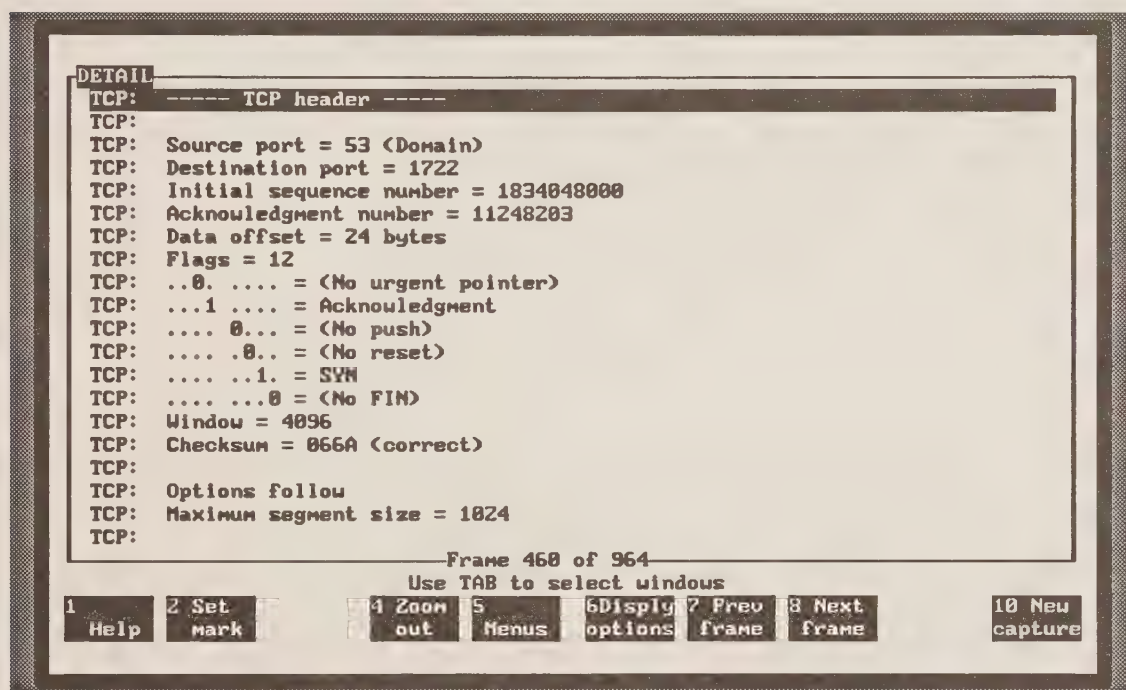
The packets sent by TCP can be one of two types. Normally, packets don't have the urgent pointer sent. These are normal data packets and may or may not have data (they be only an ACK). The other kind of packet has the urgent pointer sent (see Fig. 3-12).

All operating systems have a way of signaling the kernel that urgent data is ready to be sent: this is usually a system call or a STREAMS message. When the TCP module gets this urgent data, it puts it into a separate packet from the normal data stream and sends it off.

When the data reaches the other end, it jumps the queue for data waiting on that connection and is presented to the user, along with a signal that the data is urgent. What the data means is totally up to the user: for a Telnet session, for example, a typical piece of urgent data is "abort current command."



3-11 A TCP Header



3-12 TCP Option Negotiation



Urgent data can be sent even if the window is set at zero. It is also possible to acknowledge received data, even if the sender has set the window to zero.

### *Acknowledgment Strategies*

Flow control in an IP network is heavily dependent on the proper behavior of TCP modules. TCP modules depend on the acknowledgment mechanism to prevent a node from sending too much data onto the network. If too much data goes onto the network, the impact is on a remote host or router. If a machine in the path acting as a router or the destination host is slower, it will start dropping packets.

When a packet is dropped, it is resent, further compounding the load problem on that host. To make things worse, if enough of these situations occur, the network itself becomes congested, leading to even more problems. Likewise, a transient network problem also causes lost packets, causing more retransmissions.

It is because TCP provides a guaranteed, end-to-end delivery service that acknowledgment and retry strategies are so important. Too aggressive a retransmission of data is not good for the network. Giving up too easily, however, means lots of aborted connections.

The basic mechanism in TCP for handling these problems is the combination of an acknowledgment and a window. The acknowledgment names the highest numbered byte of data that have not been received.

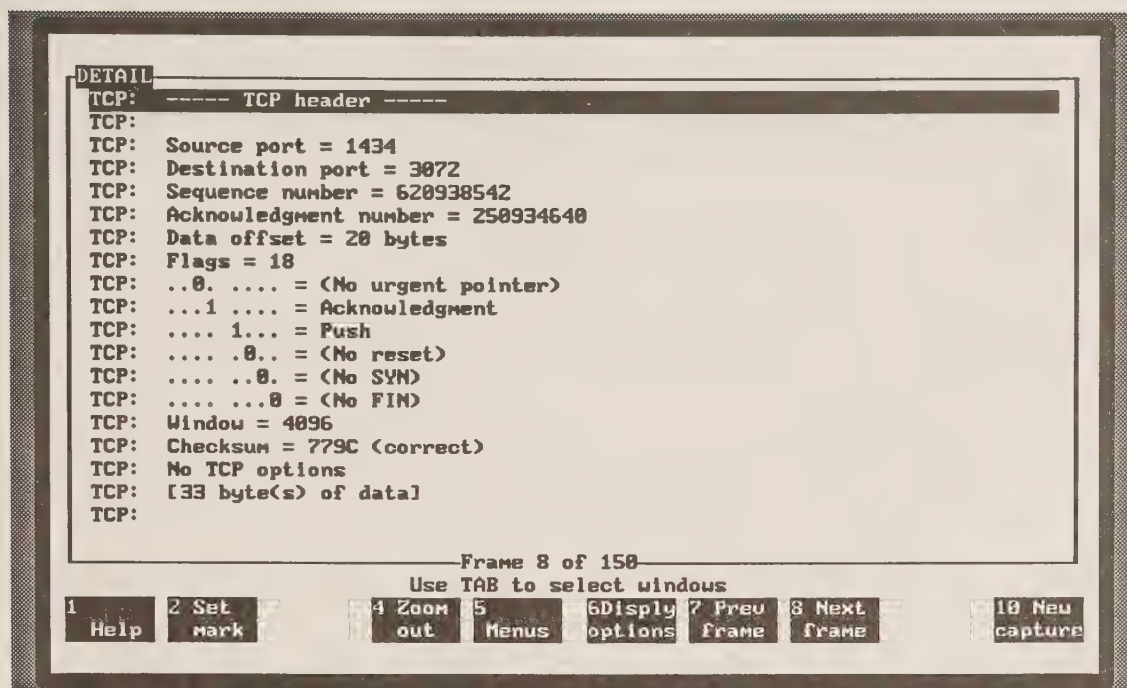
The window mechanism indicates the maximum number of bytes that the remote sender is permitted to transmit. To prevent a node from transmitting (most) data, the window value is simply set to zero. Because TCP includes an urgent notification mechanism, this means that only very small amounts of data will be accepted.

Figures 3-13 and 3-14 illustrate the concept of a window. Notice that there are two nodes exchanging acknowledgments. Let's call those nodes Bridge and Zero. In Figure 3-13, notice that Zero has sent a string of four packets, the third of which is in the detail section of the window.

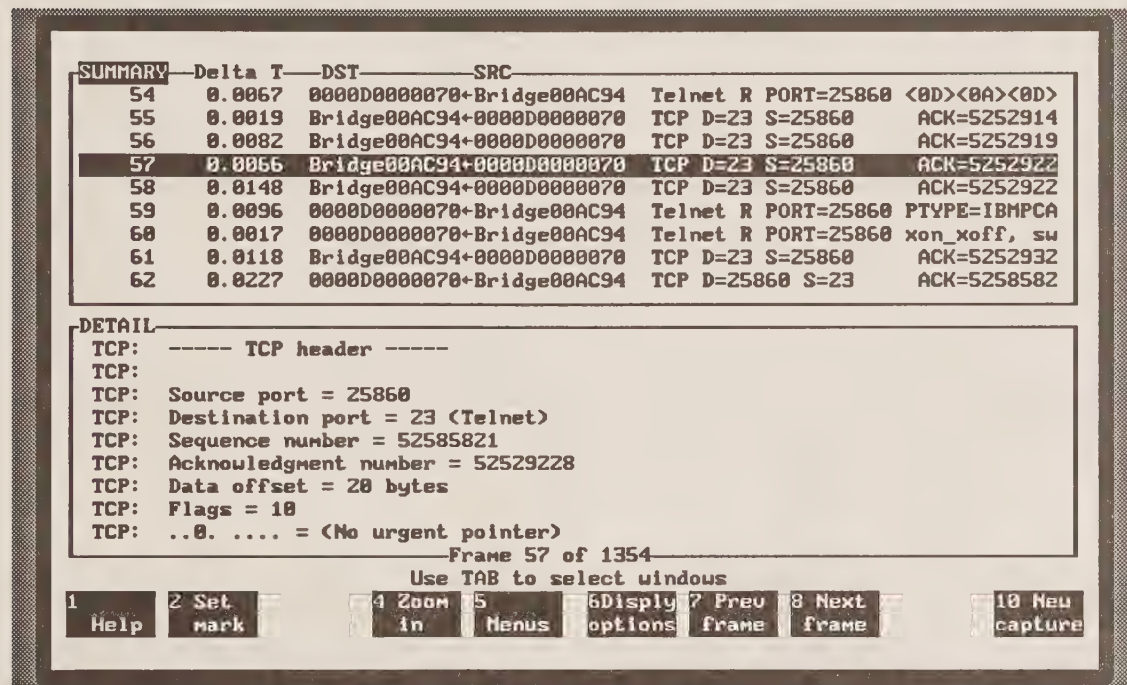
For the first three packets, there is a gradual increase in the acknowledgment packets, followed by a repeat in the fourth, indicating that a break has been reached. Figure 3-14 shows a response from Bridge. This node is acknowledging the receipt of the third data packet. The fourth has arrived but not been processed, so will be acknowledged in the next packet.

Eventually, when a steady state is reached, the connection will stabilize. It is in the middle of a transfer, when each node has a different idea of the current state of acknowledged data, that strategies are needed to properly handle data transfer.





3-13 A Push



3-14 Acknowledgment in TCP

### *Silly Window Syndrome*

The silly window syndrome was identified by Dr. David Clark of MIT in a 1982 RFC. This situation occurs when a TCP module pursues what appears to be a rational window management strategy but has the opposite of the intended results.

A node has an offered window, which is the amount of data a node is willing to accept. Often, this number is the amount of buffer space currently free at the receiver's node. While there are ways of averaging space among multiple users, let's assume for the time being that there is a single sender of data so the amount of buffer space is indeed the amount of space available for the sender.

When the sender receives the offered window, the sender uses this number to compute a different number: the usable window. The usable window is the offered window minus the amount of currently unacknowledged data.

Take a receiver that offers a window of 1000 bytes, and let's assume a maximum transmission size of 200 bytes. The sender sends five segments of 200 bytes each. Each one of these packets uses up the usable window.

Let's say the receiver gets the first 200 bytes, processes the data, and decides to acknowledge the data, sending back a window of 1000 bytes, the actual amount of space still available. After all, it seems to make sense to try to offer the largest possible window, thereby increasing throughput.

The offered window of 1000 is received at the sender, but the sender notes that there are still another 800 bytes of data in transit. Remember, five packets were initially sent and the first one arrived at the receiver. Since 800 bytes are in transit and the offered window is 800, the usable window is only 200, according to the sender's formula. So far, so good.

Now, the sender sees 50 more bytes come into the send buffers. It sends off those 50 bytes. The receiver gets the 50 bytes, and sends back an offered window of 1000. Now we have an offered window of 1000 but 850 bytes in transit, leading to a usable window of 150 bytes. If we get another 1000 bytes of data, we'll send the first 150 bytes off. Notice that we are not using the MTU of 200 bytes even though we should be: the receiver has said that 1000 bytes are available.

What has happened is that data doesn't naturally fall into the MTU of the transmission link. Occasionally, we have to send smaller segments. Acknowledging small segments leads to the artificially reduced window. Once the window has been divided, it stays small as long as data keeps flowing.

As soon as the sender stops, there is time for things to go back to normal. Everything is acknowledged and things go back to having a full usable window equal to the offered window.

During a large file transfer, however, the silly window syndrome can easily degenerate into a network full of small fragments, each one being indi-

vidually acknowledged. Of course, this clogs the network, resulting in packet losses and massive retransmissions.

This problem is solved at both the sender and the receiver nodes. At the receiver, if a node gets a small packet of data, it can artificially reduce the window offered. This means that the usable window at the sender side ends up being zero and the sender waits.

When the receiver has enough data, it removes the artificial limitation so that the sender computes a large jump in the window instead of lots of small ones. The problem, of course, is how long to keep the window closed. If it is permanently closed, there is no throughput. If it is always open, the silly window degradation occurs.

The sender can also help. The sender is able to look at the ratio of the usable window to the offered window. If this ratio is small, say 25 percent, the sender can simply refrain from sending for a while.

The basic problem here is that each segment is being individually acknowledged. Prompt ACKs are good because they let more data enter the pipeline. The goal is to balance that need with the need to prevent lots of small packets from entering the pipe.

A good general rule for a receiver is to refrain from sending an acknowledgment unless it is intended to produce an increased usable window. There are two exceptions to this general guideline. First, waiting too long causes a retransmission to occur. Waiting is thus under a timer mechanism. Second, data may be going in the reverse direction anyway, and it makes sense to piggyback the ACK on that packet.

How long to wait for an acknowledgment? If a push bit is set on the TCP data stream, this is the end of a logical record, and it will probably be followed by a pause. Acknowledgment makes sense. If there is no push bit, waiting 50 to 500 milliseconds might be appropriate.

Although the receiver can control the acknowledgment, the retransmission intervals are under the control of the sender. In theory, the sender is using the TCP delay algorithm, which uses a smoothed calculation of the round-trip delay as the basis for calculating the retransmission timer. This means that retransmissions are being adjusted based on the actual delay on the network, allowing the node to slow down during times of congestion.

### *Routing During Instability*

Gateways collectively implement a routing algorithm to find the best route between all pairs in a network. When a network or gateway fails, there will be a temporary instability in the routing database.

What happens when a TCP implementation times out during the period when the gateways are attempting to find and stabilize routes? The gate-



way gives the host feedback on good and bad routes through two ICMP messages: redirect and destination unreachable. If these messages arrive during periods of stability, they are to be believed. However, during instability they might not be. Therefore, the host should take an isolated error message with some skepticism during the transient period.

A second period where there is a problem is when the gateway in use crashes. Then, none of these messages arrives and all the datagrams are lost. Hosts need to detect this situation. Note that if the gateway is several hops away, it is the responsibility of the gateways neighboring the failed gateway to do the detection, a process which will be invisible to the sending host. This problem only occurs when the first gateway crashes.

Let's pretend the absence of the first gateway is detected. The host picks a random gateway and starts sending to it. If the random gateway isn't correct, that gateway will send back a redirect message. Another possibility, of course, is that the gateway picked at random is also down. The host will need to detect that and keep going down the list until either the message gets through or there are no gateways left to try.

## Connections and Options

Setting up a new TCP connection is the same as resynchronizing an old one: a total of three packets are sent. The first node sets the SYN bit and transmits an initial sequence number to use. The other node then replies, acknowledging the first initial sequence number and sending one of its own (see Fig. 3-15). Finally, the initiator acknowledges the target's ISN and the connection is ready to transfer data.

Since a connection is simply resynchronized, it is possible to carry user data in the initial packet, initializing the application at the same time that the transport layer sets up the connection. It is also possible to use these first few packets to set up an option negotiation.

Figure 3-15 shows that the packet includes an option to set the maximum segment size to 1024. This strategy ensures that each TCP packet will fit inside an IP packet, which in turn will fit inside a single Ethernet packet.

Setting the MTU is simply setting the size of the data that the transport layer will hand to the IP layer. IP can make its own choices about data fragmentation. The minimum size for a packet that an IP implementation must handle is 576 bytes. Even if the transport MTU is 1024 bytes, if IP decides that the packet must go over a slow serial line, it would be fragmented into two pieces.

## *Extending TCP*

With the advent of very high-bandwidth networks, TCP is starting to show its age. In the area from 300 bps to 800 Mbps, TCP works quite well. Prob-



SUMMARY	Delta T	DST	SRC	
54	0.0067	0000D0000070+Bridge00AC94	Telnet R PORT=25860	<0D><0A><0D>
55	0.0019	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252914
56	0.0082	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252919
57	0.0066	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252922
58	0.0148	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252922
59	0.0096	0000D0000070+Bridge00AC94	Telnet R PORT=25860	PTYPE=IBMPC
60	0.0017	0000D0000070+Bridge00AC94	Telnet R PORT=25860	xon_xoff, sw
61	0.0118	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252932
62	0.0227	0000D0000070+Bridge00AC94	TCP D=25860 S=23	ACK=5258582

**DETAIL**

TCP: ----- TCP header -----

TCP:

TCP: Source port = 23 (Telnet)

TCP: Destination port = 25860

TCP: Sequence number = 52529228

TCP: Acknowledgment number = 52585821

TCP: Data offset = 20 bytes

TCP: Flags = 10

TCP: ..0. .... = (No urgent pointer)

-----Frame 59 of 1354-----

Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

### 3-15 TCP Acknowledgment

lems start to occur in links where there are both very high bandwidth and a long round-trip delay. The typical examples, of course, are the wide-area gigabit links being put into place for expansion of the Internet core.

The combination of the bandwidth of a link in bits per second and the round-trip delay in seconds yields a measure of the number of bits it takes to fill a pipe. To fill the pipe, it is necessary to have a window at least big enough to permit that much data to be unacknowledged.

As the pipe starts to approach large numbers (greater than  $10^5$  bits), TCP starts to have problem. A name for this situation was proposed by Van Jacobson and Robert Braden: a "long, fat pipe network" or LFN.

An example of a simple long, fat pipe is a T1-speed satellite channel. This LFN can easily have a bandwidth times delay product of  $10^6$  bits or more. This is equivalent to 100 outstanding TCP segments of 1200 bytes each. Even terrestrial paths can have these types of problems. A 30-ms cross-country delay on a DS3 (45 Mbps) also exceeds a pipe size of  $10^6$  bits.

There are really three limitations in the basic TCP protocol.

- A window size limitation
- Cumulative acknowledgments
- Timing the round-trip delay

All three of these limitations are addressed by options that are negotiated when a connection is set up (see Fig. 3-16).

The window size limitation is the biggest one. There is a 16-bit field that includes the number of bytes that can be acknowledged. The largest possible window is thus 64 kbytes. However, since many current TCP implementations break if all 16 bits are used, a more realistic limitation is 32 kbytes. If a pipe is  $10^6$  bits, then a 32-kbyte window means that the pipe is filled at only one-quarter of its capacity.

A TCP option fixes this by allowing an “implicit scale factor” to be established when the connection is set up. This option allows the user to multiply the offered window by the scale factor to arrive at the true window size.

The second problem is cumulative acknowledgments. In the default TCP acknowledgments, if a particular piece of data has a problem, the sender is forced to resend all packets up to that point that were not acknowledged, even if the other packets had no problems.

A TCP option is added to handle the problem of selective acknowledgment. A selective acknowledgment (SACK) is used in two phases. First, when the connection is set up, both sides have to agree to use SACKs. Second, as an option in future TCP packets, the receiver includes a SACK which specifies the length and the offset of the data being acknowledged.

The third basic problem in the default TCP is the problem of measuring the round-trip timing (RTT). Most TCP implementations use the RTT as the basis for their retransmit timers. In order to properly react to the true network congestion, we need an accurate RTT.

In most implementations, RTT is measured between the time a packet left and the time it is acknowledged. Since there are cumulative acknowledgments in TCP, most RTT timers are based on a single measurement per window. While this is fine with small windows, it can cause problems when the windows become very large. The solution to measuring the RTT is an echo option. The data in the echo option is immediately sent back, allowing a measure of the round-trip delay to be obtained at negligible cost.

In the basic TCP specification, there is no well-defined concept of options being sent at any time. In fact, there is really only one option, to set up the MTU at the time a link is synchronized. When defining additional options, one of the goals is of course to keep the current implementations working.

To achieve this, all the TCP options must be negotiated at the beginning. They can't be added once a connection has been set up. This is under the theory that a poor existing implementation would probably ignore extra additional options on the first packet but would crash if it saw them on additional packets. Once agreement is achieved at the beginning, the two communicating TCP modules can use the extensions.

Name	#	Length	Description
End of Option List	0	—,	
No-op	1	—	No operation.
Maximum Segment Size	2	16 bits	Must be sent only in the initial connection request.
Window Scale Option	3	3	Data is a shift count parameter which represents the number of bits by which the receiver right shifts the true window-receive value.
SACK Permitted	4	2	Selected ACKs permitted. Must be sent in the initial SYN.
SACK Option	5	variable length	Contains a list of blocks of contiguous sequence space that have been received. Each block is based on two 16-bit integers: the relative origin (relative to the last ACK number field in the TCP header which is the left side of the window) and the block size. Note there are 44 bytes available for TCP options. Therefore, this option allows selective ACKs of 10 blocks. Note that use of other options cuts down on the space available for SACKs. If scaling is in effect, SACK fields are scaled by the same factor as the window.
TCP Echo Option	6	4 bytes	Data to be echoed. When used for round-trip timing (RTT) measurement, field will contain a timestamp.
Echo Reply	7	4 bytes	

### 3-16 TCP Option Codes

#### *Source Quench and TCP*

The approved weapon against congestion is the ICMP Source Quench Message. If a gateway or host is getting data too fast, it sends this message to the host that sent the original data. When a host receives this message, it is expected to take actions to reduce the amount of traffic it sends to the system that sent the source quench.

There are really two main causes of congestion:

- Tinygrams
- Lack of support for source quench

Tinygrams contain very little data and are typically generated by character-at-a-time applications like Telnet. TCP can reduce tinygrams by refusing to send new data until previous data has been ACKed.



Another problem is not adapting when you get a source quench. Under normal operation, a TCP module will see that there is congestion and reduce the amount of the offered TCP window that it will fill. If a remote TCP is currently offering a window of 4096 octets and the local TCP receives a source quench, it starts to act as though the window size is actually 1024 (even though the remote continues to advertise 4096). After a while (count the number of ACKs received, for example), the local TCP resumes using the real window size.

Two other schemes are Raj Jain's congestion control using timeouts at the end-to-end layer (CUTE) and Van Jacobson's slow start method. With both these schemes, instead of shrinking the window until congestion is reduced, the window is slowly increased until congestion occurs. When congestion occurs, the window is shrunk to the minimum size and again allowed to grow slowly until congestion occurs again. Both of these schemes use the number of lost packets to detect that there is congestion present instead of using the source quench method. The logic here is that it is better to slowly build up load to achieve a steady state than to hammer a circuit to force source quench. The later approach is conducive to congestive collapse of an internetwork.

## TCP and OSI

Another transport service provider is the ISO Transport Protocol, a series of five different transport protocols that each provide a different level of service. TP Class 4 (known as TP4) provides the highest level of service to an OSI application and is functionally equivalent to the TCP protocol (see Fig. 3-17).

In the OSI world, the TP4 transport module would use one of two different network services, the Connection-Oriented Network Service (CONS) or the Connectionless Network Service (CLNS). CLNS is virtually identical to IP, and TP4 is virtually identical to TCP.

Sun workstations would access these services in one of two ways. A native implementation of the OSI protocols, together with the applications, can be purchased. Alternatively, a SunLink gateway can be put on the network. This SunLink gateway is a server, just like any other server. Workstations can connect to this server and then access the basic OSI applications such as FTAM. In this sense the gateway is routing at two different levels: it connects the transport and network layers of the two environments together, but it also connects two applications together.

## *TP0 over TCP*

Need to run OSI applications but don't have an OSI network? Several Internet application designers felt that they wanted some of the services pro-



Category	TCP	TP4
Data Transfer	Streams	Blocks
Flow Control	Bytes	Segments
Error Detection	Checksum	Checksum
Addressing	16-bit port number	Variable TSAP-ID
Interrupts	Urgent pointer	Expedited data
Security	11-byte IP field	Variable TP, IP fields
Precedence	3-bit IP field	16-bit TP, IP fields
Datagrams	UDP	ISO 8602
Disconnect	Graceful	Abrupt
Source: <i>Transport Protocols for Department of Defense Data Networks: Report to the Department of Defense and the National Bureau of Standards</i> , National Academy Press, February 1985. Table reprinted in David M. Piscitello and A. Lyman Chapin, "TCP and TP4: Moving Forward,," pp. 2-9, <i>ConneXions</i> , Vol. 4, No. 9, September 1990.		

### 3-17 TCP and TP4 Comparison

vided in the upper OSI layers, particularly the session and presentation layers, but didn't want to wait for OSI networks.

Since TCP and TP4 both provide the same basic service—guaranteed error-free delivery of data—there is no reason why the stacks can't be spliced. The TP0 transport protocol assumes an underlying connection-oriented network service (e.g., X.25). Splicing TP0 on top of TCP creates not only a connection-oriented network service but a reliable one as well, yielding the same semantics as OSI TP4.

This procedure is defined in RFC 1006. TCP port 102 is reserved for its user: TP0. There are a couple of minor differences between the protocols that must be resolved. For example TCP has a concept of ports, but TP doesn't. Instead of waiting for an incoming indication on a port, the remote TCP/IP-based OSI application must post a listen.

The primary user of RFC 1006 is the ISO Development Environment (ISODE) developed by Marshall Rose of Performance Systems International (see Fig. 3-18). ISODE implements the major OSI layers from the session layer through the application layer. It has some applications built in, most notably an implementation of the X.500 directory. Other application developers also use ISODE to develop programs (again, most notably implementations of X.500).

The lesson to learn from ISODE is that network architectures are not immutable: protocols can be pulled from various architectures as needed. This is an important lesson since many people looked to a "transition" to OSI, whereas ISODE shows that coexistence is indeed possible.

## UDP

The user datagram protocol is a simple alternative to TCP. It is built on IP and provides little value added. In effect, UDP is simply a user interface to IP. In particular, guaranteed delivery and protection against duplicate packets are not features of UDP.

Why would an application wish to use UDP? It has a few advantages over TCP. First, because it is not connection-oriented it is possible to use the underlying IP multicast service to send a single UDP packet to multiple hosts. Second, UDP is less resource intensive than TCP. Because there is no need to set up a connection or to assure the delivery of data, it will use fewer resources. If the environment is something like an Ethernet, with a low probability of packet loss and duplication, UDP may be appropriate. Third, UDP is less complex to implement; hence the resulting code is smaller. This was a factor in a protocol like SNMP which has to function in devices with limited resources. Fourth, UDP lends itself to implementing a service as NFS which provides support for large numbers of clients.

In any case, applications that use UDP have to worry about the delivery of data. Some applications simply don't worry about it: if a packet doesn't arrive, nothing happens. Other applications use a request-response scenario with retransmitted packets in case of a failure to respond to a request. When a request is sent, the protocol indicates that a response will be received at some point. If the retransmission timer expires, the request is simply sent again. This is the method that the Network File System uses when it is built on top of UDP (see Fig. 3-19).

The UDP packet format is quite simple. The packet has 2-byte source and destination ports just like TCP. The source port is optional and can be zero. In addition to addresses, the UDP packet has a length indicator and a checksum, followed by data. The length includes the header and data, so the minimum valid value is 8. The maximum is 8 kbytes. The checksum is the 16-bit one's complement of the one's complement sum, although a value of 0 is allowed, which means that the transmitter generated no checksum.

## RIP

We move now from transport protocols to routing control. There are two classes of routing control protocols: interior protocols are used within a routing domain; exterior protocols are used between separate routing domains. We start first with interior protocols then move on to discuss exterior protocols such as EGP.

The Routing Information Protocol (RIP) is an example of an interior routing protocol, and it is commonly used on networks that come from a Berkeley 4.2 Unix ancestry, including many Sun networks. RIP is built on top of the UDP transport protocol.

**DETAIL**

```
TCP: ----- TCP header -----
TCP:
TCP: Source port = 1082
TCP: Destination port = 102 (ISO)
TCP: Sequence number = 1065688
TCP: Acknowledgment number = 38800
TCP: Data offset = 20 bytes
TCP: Flags = 18
TCP: ..0. .... = (No urgent pointer)
TCP: ...1 .... = Acknowledgment
TCP: .... 1... = Push
TCP: .... .0.. = (No reset)
TCP: .... ..0. = (No SYN)
TCP: .... ...0 = (No FIN)
TCP: Window = 4096
TCP: Checksum = A8BA (correct)
TCP: No TCP options
TCP: [6 byte(s) of data]
```

**ISO\_DE: ----- ISO Development Environment -----**

```
ISO_DE:
ISO_DE: Multi-frame TPDU: frames 10, 11, 12
ISO_DE: Version = 3
ISO_DE: Packet length = 132
ISO_DE:
ISO_TP: ----- ISO Transport Layer -----
ISO_TP:
ISO_TP: Header length = 2
ISO_TP: TPDU type = F (Data)
ISO_TP: EOT: Last TPDU of a sequence
```

**ISO\_SS: ----- ISO Session Layer -----**

```
ISO_SS:
ISO_SS: SPDU type = 13 (Connect)
ISO_SS: Length of SPDU parameter field = 123
ISO_SS: -- Connection identifier parameter group (length = 28)
ISO_SS: Calling SS-user reference = <1407>kurt380
ISO_SS: Common reference = <170D>881109193935Z
ISO_SS: -- Connect/Accept item parameter group (length = 12)
ISO_SS: Protocol options = 00
ISO_SS: .... ..0 = Not able to receive extended concatenated SPDUs
ISO_SS: Maximum TSDU size = 65528 (Initiator to responder)
ISO_SS: Maximum TSDU size = 65528 (Responder to initiator)
ISO_SS: Version number = 1
ISO_SS: Session user requirements = 000Z
ISO_SS: .... 0... .... = No symmetric synchronize
ISO_SS: .... .0.. .... = No typed data
ISO_SS: .... ..0. .... = No exceptions
ISO_SS: .... ...0 .... = No capability data
ISO_SS: .... .... 0... .... = No negotiated release
ISO_SS: .... .... .0.. .... = No activity management
```

Frame 10 of 203

Use TAB to select windows

1 Help	2 Set mark	4 Zoom out	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	------------	---------	-------------------	--------------	--------------	----------------



SUMMARY	Delta T	DST	SRC	
2	0.0283	DEC	029487+Intrln0027C0	NFS R OK (4096 bytes)
3	0.0077	DEC	029487+Intrln0027C0	UDP continuation ID=6184
4	0.0070	DEC	029487+Intrln0027C0	UDP continuation ID=6184
5	0.0150	Intrln0027C0+DEC	029487	NFS C F=4E0E Read 4096 at 65536
6	0.0110	Intrln0027C0+DEC	029487	NFS C F=4E0E Read 4096 at 69632
7	0.0237	DEC	029487+Intrln0027C0	NFS R OK (4096 bytes)
8	0.0084	DEC	029487+Intrln0027C0	UDP continuation ID=6440
9	0.0077	DEC	029487+Intrln0027C0	UDP continuation ID=6440
10	0.0152	Intrln0027C0+DEC	029487	NFS C F=4E0E Read 4096 at 73728
11	0.0013	DEC	029487+Intrln0027C0	NFS R OK (4096 bytes)
12	0.0079	DEC	029487+Intrln0027C0	UDP continuation ID=6696
13	0.0070	DEC	029487+Intrln0027C0	UDP continuation ID=6696
14	0.0212	DEC	029487+Intrln0027C0	NFS R OK (4096 bytes)
15	0.0082	DEC	029487+Intrln0027C0	UDP continuation ID=6952
16	0.0074	DEC	029487+Intrln0027C0	UDP continuation ID=6952
17	0.0050	Intrln0027C0+DEC	029487	NFS C F=4E0E Read 4096 at 77824
18	0.0129	Intrln0027C0+DEC	029487	NFS C F=4E0E Read 4096 at 81920
19	0.0176	DEC	029487+Intrln0027C0	NFS R OK (4096 bytes)
20	0.0078	DEC	029487+Intrln0027C0	UDP continuation ID=7208
21	0.0077	DEC	029487+Intrln0027C0	UDP continuation ID=7208

1 Help

2 Set mark

4 Zoom out

5 Menus

6 Display options

7 Prev frame

8 Next frame

10 New capture

Use TAB to select windows

### 3-19 UDP Continuation

Although RIP has some problems, it is useful for small networks with simple topologies. RIP is an example of a distance vector algorithm. The distance is based on the concept of hops: each hop typically being a single subnetwork. It is possible to give slow links a higher hop count. RIP limits the total distance of a node to 15 hops away: 16 is considered infinity and thus unreachable.

The reason we call this a distance vector algorithm is that we don't know about all the different links we will take. All we know is the neighbor to use (the beginning of the vector) and the total distance.

Each router will periodically give RIP packets to its neighbor routers. That packet will list all the networks and hosts that the router can reach. A given router will receive RIP packets from several routers. By comparing the distance to a given host for each of the different routers, we can determine which one is the best path.

Notice that we depend on each router to tell us about the best path it knows about. We, in turn, will tell other routers about the best paths that we know about. This is like telling our neighbors about the world as we know it. During times of stability, this works fine. However, in an unstable topology, it may take a while for nodes to sort out which path is the better path.



Despite their potential limitations, distance vector algorithms are still widely used. For one thing, they are widely implemented. Distance vector algorithms were used as early as 1969 in the ARPANET. RIP's ancestry, however, is from XNS RIP. RIP is very similar to XNS RIP, with general addresses replacing XNS addresses and with routing updates limited to one every 30 seconds.

RIP is based on the concept of fixed metric. The metric is a hop, and the "cost" for each particular link is fixed. This means that routes based on real-time parameters such as measured delay, reliability, or load cannot be chosen.

The maximum distance limit of 15 also imposes some real limits in assigning costs. Let's say there are FDDI networks operating at 100 Mbps and 9.6-kbps phone lines. Since the phone line is 10,000 times slower than the FDDI link, it would make sense to have it cost 10,000 times as much—tough to do with a cost space of 15.

RIP is based on five pieces of information maintained on a router:

- Destination IP address
- Gateway IP address
- Physical network interface to reach gateway
- Current metric of distance
- Time since this entry last updated

This information is basically the forwarding database. Given a network address on a different subnetwork, a router looks in the database and sees which gateway to use. If the router gets a RIP packet with a better metric, the current entry in the database is replaced with the new one.

What if the route got worse? If the gateway that was going to be used for the packet says the cost got worse, it should be believed and the metric is updated. If another node has a worse route, it is ignored.

What happens when a router crashes? In the RIP world, neighbors are updated every 30 seconds. If a neighbor has not sent an update for a particular route for 180 seconds, the route is marked as invalid. As soon as a particular network is known to be unreachable (at least from the perspective of a particular router), the metric is set to 16 (infinity).

A potential problem with this scheme occurs when two nodes each think the other might be the right route. To prevent this problem, a technique is used called the split horizon. This mechanism says that updates are only sent to neighbors from which that particular update was not received.

A version of this is split horizon with poisoned reverse, which includes all the routes in the updates, but puts the metric to infinity. Take two nodes, A and B. Both are on the same subnetwork and both can reach D. When A sends a routing update to B it says D exists with a cost of infinity: B should not use A to send to D since it can do so directly. This reversed

poisoning is a simple way of breaking routing loops. If two routes are pointed at each other, setting the cost to 16 breaks the loop immediately (whereas if the route weren't broadcast, it would be necessary to wait for a timeout to eliminate it).

Split horizon with poisoned reverse prevents routing loops with two gateways, but it is still possible to have patterns of three gateways in mutual deception. The only way to stop this is to get the infinite route information out to all three of them. The way this is handled is to trigger an update when a network becomes unreachable.

RIP is based on the UDP transport protocol using port 520. All routing update messages originate on this port, and all unsolicited routing update messages originate and go to this port. A response to a request is sent to the port from which it came.

It is possible for some nodes to be a silent RIP implementation, simply listening to the network. A host might do this, or a gateway that has just initialized would also listen for a period to prevent sending neighbors unreliable information.

RIP allows the transport of routing information for different network layer protocols, but currently it only has an address family defined for the Internet Protocol. Given RIP's reputation as a user of bandwidth and the inability to scale to large networks, RIP is unlikely to be expanded to include other address families.

RIP packets are limited to a maximum datagram size of 512 octets, thus letting the packet fit into the 576-octet packet size that all Internet hosts are required to handle. This ensures that routing packets are not fragmented. Multiple packets can be sent for long routing tables.

An entry in a RIP packet is identified by an address. The address can refer to an individual host or a network, or a special code can be used to indicate the "default address." If every host on a network is accessible through a particular gateway, the hosts don't need to be listed individually. In fact, some implementations don't support host routes and drop any that are received.

Network numbers can also be subnetwork numbers. The basic assumption is to start with the most specific entry and then become more general. First look for host entries, then subnet entries, then network numbers, and finally the default route.

The way to interpret a network number is to apply the local subnetwork mask. Assume you have subnet mask of 255.255.255.0. The 255 is the decimal representation of 1111 binary, meaning that all four bits of that portion of the mask have been turned on. Based on this mask, 128.6.0.0 is a network number, 128.6.4.0 is a subnetwork number, and 128.6.4.1 is a host address. Note that subnetwork numbers don't mean anything unless you have the subnetwork mask for that network. Therefore, a host shouldn't

send subnetwork addresses to another host unless it is part of the same network. Sending it to another network is silly.

Filtering of subnetworks is carried out by gateways on the border. On the interior, each subnet is a separate net. The border gateway would only send a single entry for the network as a whole. Using the same logic, a border gateway shouldn't include single host information when talking to other networks. Instead, the border gateway should advertise a single network entry.

An address of 0.0.0.0 is the default route. The default route is used to direct all packets that don't match an entry in the routing table to a "smarter" gateway. How do we determine which gateway should be the default? One rule is that any gateway speaking an exterior gateway protocol such as EGP is a default. If there are multiple EGP gateways, the metric would make one the "preferred default." Typically, each autonomous system (AS) has its own preferred default gateway and the 0.0.0.0 routes shouldn't leave the boundary of the AS.

RIP is based on the concept of timers. Every 30 seconds, a node should send a complete RIP response to every neighbor gateway. In addition to a timer for neighbor gateways, the RIP module maintains two timers for each route:

- A timeout value
- A garbage collection timer

When a route expires (the timeout value expires), the route is no longer valid. However, the entry is retained for a short period of time so that it will be included in the next update packet. When the garbage collection timer expires, the route is deleted.

The value for the timeout timer is 180 seconds. When timeout occurs, or when an entry is received with a metric of 16, the garbage collection timer is set to 120 seconds. If the metric is not already at 16, it is set to 16. Note that when an entry is changed, this also sets the flag. The flag tells the output process to trigger a response, allowing neighbors to learn this new information.

### *RIP Requests*

Normally requests are sent as broadcasts. If the request comes from port 520, it is meant for all active RIP processes, and the silent ones should stay quiet. If the request comes from a port other than 520, even silent processes should respond.

Requests are processed entry by entry. If there are no entries, no response is made. A special entry is an address family of 0 (unspecified) and a metric of 16, which means send the whole routing table. Except for that case, processing is simple. The RIP module goes down the request line by



line and compares the entries to the forwarding database. If there is a route in the forwarding database, then the metric is put into the packet. Otherwise, a value of 16 is put in. When all the responses have been filled in, the datagram is sent back to the port from which it came.

Note that when a remote node asks for specific entries, a RIP module answers with that specific information. However, if the whole table is being processed, the normal rules of subnet hiding and split horizon with poisoning apply. The idea is that if a specific host or network is asked for, this is a diagnostic request (probably by a human) and therefore one should tell the system what it asked for. On the other hand, if the remote system is booting and thus sending a RIP request, it is necessary to put in the subnet hiding and poisoning features.

### *Response Processing*

Responses can be received for three reasons:

- Response to query
- Regular update
- Triggered update because of changed metric

Processing of responses is always the same. If a response is not from port 520, it is ignored. If the response IP source address is not a valid, directly connected neighbor, it should likewise be ignored. It is also a good idea for a RIP module to check and see if the response is from itself, as it is possible on a broadcast network for the sending node to receive its own responses.

Once the initial sanity checks are done, the individual entries are checked. If an entry has a value greater than 16, it is ignored. Next, the family ID is checked to see if it is relevant. Then, the address itself is checked to see if it is appropriate. If the address is on network 127 (the loopback network), it is ignored. Also ignored are broadcast addresses and, if the node doesn't support them, host routes.

Next, the metric in the RIP field is updated by adding the cost of the network from which the network arrived. If the total is greater than 16, 16 is used. At this point, the address in the RIP entry is looked up in the routing database. If the address is not present, it is added to the routing database (in most cases). Infinite metric routes are not, of course, added. Neither are host-specific routes if there is already a subnet route as good or better in the database.

If the entry is added, four steps are involved:

- Add the address and set the distance.
- Set the gateway as the source of the datagram.
- Initialize the timeout.
- Set the route change flag and send a signal to trigger an update.



If there is already an existing route, then the gateway specified in the forwarding database is compared with the gateway listed in the RIP response. If the two are the same, then the information in the routing database is updated and the timers are reset. If the new metric is 16, then the garbage collection process is initialized and an update triggered. If the gateways are different and the new metric is better, then it is added to the database.

If the gateway is different and the metric is the same, one would normally ignore it. However, in the 4.3BSD implementation of the RIP route daemon (routed) the information is used. If the existing route is about to time out, it makes sense to use the gateway that has just arrived (and is thus good for a while). A good rule of thumb is that if the timeout is at least halfway to its expiration point, it makes sense to switch to the equivalent route.

Once all the new entries are merged into the database, the RIP module has to inform its neighbors. Packets are sent either in response to a request, every 30 seconds, or by a triggered update when an entry is updated.

Triggered updates require special handling to avoid bombarding the network. When a triggered update is sent, a timer is set that has a duration picked at random between 1 and 5 seconds. During that time, no more updates can be sent. Note that a triggered update is always suppressed if a regular update is due by the time the triggered update timer expires.

The Berkeley routed daemon has two administrative controls that are added. First, a neighbor list is maintained that contains a list of the neighbors for a particular host. If the neighbor list is active, the RIP module will accept response messages only from hosts on its list of neighbors. A second addition allows or disallows specific destinations. A list is associated with a particular interface and an incoming or outgoing direction. The RIP module can either filter out or filter in a specific list; all other destinations are disallowed or allowed.

Figure 3-20 shows the format for a RIP packet. The two commands are the request and the response. The variable fields consist of a target address, the IP address of a gateway, and the cost to reach the destination. Figure 3-21 shows a typical packet.

## OSPF and IS-IS

The Open Shortest Path First (OSPF) is one of two protocols being developed to handle very large routing domains. It is an Internal Gateway Protocol (IGP) for the TCP/IP environment, but it is significantly more sophisticated than protocol such as RIP.

Based on a link state routing algorithm, OSPF adapts much more quickly to topological changes without flooding the network with excessive routing traffic. In a link state algorithm, instead of telling our neighbors about the

Fixed Fields	
Command	1 byte
	1: request (send all/part of routing table)
	2: response (response to request or poll)
	3: traceon (obsolete)
	4: traceoff (obsolete)
	5: reserved (used by Sun Microsystems)
Version	1 byte
Must be zero	2 bytes
Variable Fields	
Address family ID	2 bytes
Must be zero	2 bytes
IP address	4 bytes
Must be zero	4 bytes
Must be zero	4 bytes
Metric	4 bytes

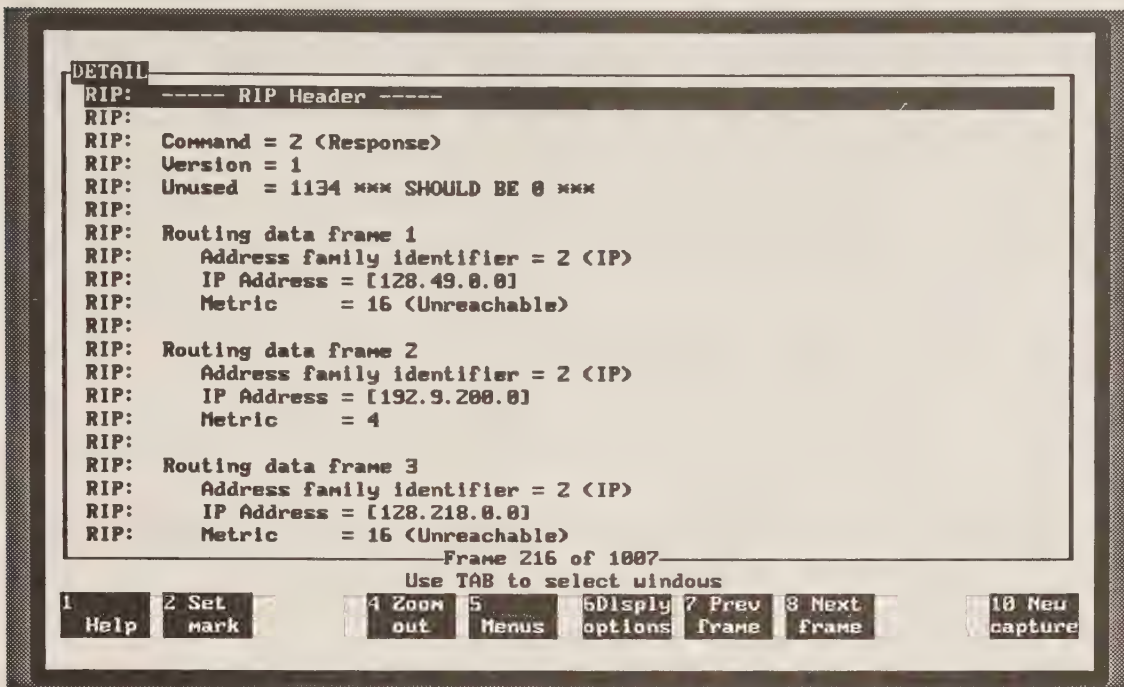
**3-20** RIP Packet Format

world, we tell the world about our neighbors. In other words, everybody in a routing domain knows about the status of a given link, meaning that each router can calculate the best route itself instead of depending on (possibly outdated) information from neighbor routers. Since link states are used, there is no reason why there can't be more than one minimum cost path to a destination, allowing equal-cost multipath routing.

A 16-bit metric is maintained for each type of service on an interface. While this doesn't mean that a different metric will be used for each service, it at least provides an architecture that allows more sophisticated routing control.

Along with IS-IS (the OSI equivalent protocol), OSPF shares the characteristics that most link state routing protocols have. First, over time every router ends up with an identical view of the network: an identical routing database. Since each router has the complete topology, routing tables can be built independently, avoiding the oscillation effect that occurs in a vector-based environment.

OSPF is based on the concept of areas. Collections of nodes (often collections of networks) are collected into an area. An area router is able to find any node within that area. This doesn't mean that the router is on the same subnetwork as the destination node, only that it has routing information to reach the target.



3-21 A RIP Packet

Other routers handle routing in between areas. If an area router sees an address outside its area, it hands the packet off to the interarea router. That router knows how to get to other areas, where the packet is handed off to an intraarea router. Division of the routing problem into hierarchical domains is a useful technique for building very large networks.

OSPF distributes its externally derived routes (e.g., EGP routes) independently from internally derived routing information. We assume that the nodes inside a routing domain (an OSPF area) have better information than the nodes outside the domain.

The basic operation in OSPF is to periodically send out a description of a router's neighbors. This is known as a link state advertisement (LSA) and is flooded throughout the router's domain. LSAs mean that the databases stay in sync. Periodically each router runs an algorithm on the database to calculate the shortest-path tree. This tree is then used to build the forwarding database (routing table).

On multiaccess networks (such as an Ethernet) there are a designated router and a backup designated router. Each router on the broadcast-type network sends an LSA to the designated router, which in turn generates an LSA on behalf of the network.

Discovering neighbors is done using the OSPF Hello Protocol. When a router sends a hello, the packet includes a list of all the routers from which



it has recently received a hello packet. A router thus knows that the packet has been seen by a neighbor by checking that neighbor's hello packet.

On broadcast networks, hellos are sent as multicasts. This means that neighbors are dynamically acquired on networks like Ethernet. The Hello Protocol is also used to elect a designated router. Once a designated router is elected, there is an adjacency between the designated router and all the other routers. On point-to-point links, there is always an adjacency.

Database synchronization is done by adjacent routers. They send each other a summary list of the database of link state packets. Each router can then build a set of requests for the ones it doesn't have.

A router in two areas keeps two databases. Routers on the border thus have a more demanding role. An area is like a generalization of the subnet concept. A subnet is inside an area, and areas are inside a network (also known as a routing domain).

Routing within an area is at the low level of the OSPF routing hierarchy. Each area is connected to the others via the spine of the AS, known as the OSPF backbone. The backbone is basically the collection of routers that belong in multiple areas (border routers).

The highest level of the OSPF hierarchy (one is within an area, the second is on the backbone of the AS) is the intra-AS routing. At this level, protocols like EGP or the Border Gateway Protocol (BGP) are used to exchange route information. External route information is then flooded throughout the entire AS.

## Exterior Gateway Protocols

The Exterior Gateway Protocol (EGP) is a way to exchange network reachability information between two neighbors on the border of a router domain. This can be within an autonomous system, but more often it is used to exchange routing information between two different administrative domains.

EGP defines a set of basic messages that are exchanged between peers (see Figs. 3-22 and 3-23). There are three pieces to the EGP protocol:

- Acquire neighbors
- Monitor neighbor reachability
- Exchange net-reachability information

The protocol is based on periodic polling using Hello and I-Heard-You (IHU) message exchanges. There are also commands to periodically poll a neighbor and to solicit updates.

All EGP commands and replies have a sequence number. The last sequence number received in a command from a particular neighbor is used for all replies and indications to that neighbor until a different sequence



EGP Message types		
Request		Request acquisition of neighbor and/or initialize polling variables.
Confirm		Confirm acquisition of neighbor and/or initialize polling variables.
Refuse		Refuse acquisition of neighbor.
Cease		Request deacquisition of neighbor.
Cease-ACK		
Hello		Request neighbor reachability.
I-H-U		Confirm neighbor reachability.
Poll		Request net-reachability update.
Update		Net-reachability update.
Error		
Fixed Parameters		
P1	30 seconds	Minimum interval between successive hello commands.
P2	2 minutes	Minimum interval between successive poll commands.
P3	30 seconds	Minimum interval between request or cease command retransmissions.
P4	1 hour	Interval during which state variables are maintained in the absence of commands or responses in the down and up states.
P5	2 minutes	Interval during which state variables are maintained in the absence of responses in the acquisition and cease states.
P4 and P5 are only used if the abort-timer option is implemented.		

### 3-22 Exterior Gateway Protocol

number is received from that neighbor. The management of sequence numbers is implementation-dependent, but the recommended method is to maintain a different one for each neighbor and increment the value just before a poll command.

The EGP divides the Internet into the “core” AS and many “stub” ASs. The core AS acts as a hub for passing routing information between different stubs. Typically, a regional network would be the stub as far as the NSFnet core is concerned. A local network might then (if it is using EGP) be a stub and the regional network would be a core.

Neighbor acquisition is based on a two-way handshake by which you agree to conduct EGP by exchanging request and confirm messages. Acquisition is terminated by cease and cease-ACK messages.

EGP Header Format		
EGP Version Number	1 byte	Current = 2
Type Field	1 byte	
Code	1 byte	Message subtype
Status	1 byte	Message dependent status info
Checksum	2 bytes	
Autonomous System #	2 bytes	
Sequence #		
Variable Fields		
Neighbor Acquisition Message		
Type		type = 3
Code		0: request command
		1: confirm response
		2: refuse response
		3: cease command
		4: cease-ack response
Status Field		0: unspecified
		1: active mode
		2: passive mode
		3: insufficient resources
		4: administratively prohibited
		5: going down
		6: parameter problem
		7: protocol violation
Hello Interval	2 bytes	In seconds
Poll Interval	2 bytes	In seconds
Neighbor Reachability Message		
Type		type = 5
Code		0: hello command
		1: IHU response
Status		1: up state
		2: down state
Poll Command		
Type		type = 2
Code		code = 0
Status		0: indeterminate
		1: up state
		2: down state
IP Source Network	1-3 bytes	

Update Response/Indication		
Type		type = 1
Code		code = 0
Status		0: indeterminate
		1: up state
		2: down state
		128: unsolicited message bit
Number of Interior Gateways	1 byte	
Number of Exterior Gateways	1 byte	
IP source network		
Gateway 1		IP address (without network number)
Number of Distances		
Distance 1		
Number of Nets		
Net 1,1,1		
Net 1,1,2		
Distance 2		
Number of Nets		
Net 1,2,1		
Net 1,2,2		
Gateway N		IP address (without network number)
Number of Distances		
Description		Field descriptions for each network
Number of Gateways		Number of interior/exterior gateways in this message.
IP Source Address		The IP network number for which we are providing reachability information (1–3 octets).
Gateway IP address		Without the network number (1–3 octets).
Distance		Depends on the autonomous system architecture.
Nets		The IP network number reachable via this gateway.



Error Response/Indication		
Type		type = 8
Code		code = 0
Status		0: indeterminate
		1: up state
		2: down state
		128: unsolicited message bit
Variable Field	2 bytes	
Reason		0: unspecified
		1: bad EGP header format
		2: bad EGP data field format
		3: reachability info unavailable
		4: excessive polling rate
		5: no response
Bad Data		Three 32-bit words of EGP header for which this error occurred.

3-23 (Cont.) EGP Packet Formats

Neighbor reachability is then determined with the Hello and IHU responses. The gateway sending the hello is in active mode, the gateway that responds is in passive mode. After that, network reachability is determined via both polls and updates. There is a minimum 2-minute separation between messages to prevent flooding a network with EGP packets.

A typical router speaks some interior gateway routing protocol in addition to EGP. In a typical example, the Berkeley 4.2 EGP implementation, the router keeps two sets of tables. One has exterior routes learned via EGP exchanges. The second table keeps track of interior routes learned by some other protocol (e.g., RIP).

When a new EGP update is received, this information is put into the exterior routing table. The normal rules apply: always update a packet in which the advised gateway is the same as the existing one, and also do an update if there is a different advised gateway with a lower metric. After some period of time, an un-updated route is deleted. Generally, any route that is not updated within 3 times the maximum poll interval should be deleted.

There is one exception to this, the default route. Most routing protocols maintain a default route which is used whenever a network is totally unknown. The basic idea is that this router doesn't know the destination, but some bigger, smarter router might have an answer.

The worst that can happen is that the neighbor router also does not know the destination and simply returns an ICMP destination unreachable mes-



sage. Slightly better would be a redirect message from ICMP. Even better is that the neighbor does in fact know what to do and simply sends the packet on its way.

## Routing Between NSFnet and Milnet

An interesting case study of routing between separate autonomous systems can be found in the way that the NSFnet, the core to the Internet, and the two military networks (Milnet and ARPANET) are connected.

NSFnet is the backbone network which connects regional midlevel networks. In turn, those regional networks have connections to autonomously administered campus and research center networks.

The two military networks, Milnet and ARPANET, are used by the Department of Defense for their various unclassified needs. However, there is a lot of interaction between the Internet and the ARPANET research centers.

Until mid-1989, there were three gateways that resided in mid-level networks. The gateways kept one EGP session with the NSFnet NSS core routers and another one with the Defense Data Network (DDN) Mailbridge.

The way it worked for inbound traffic from the DDN to NSFnet was that each NSS accepted the DDN advertised routes at a different metric. This made one NSS a “hot” link to the DDN, with the other two (with higher metrics) acting as hot backups.

For outbound traffic from the NSFnet to the DDN, each NSS router maintained an EGP peer relationship with its gateway on the midlevel network. Via EGP, it announced a certain set of NSFnet-connected networks to its peer. The NSFnet gateway on the midlevel network would then redistribute that information to its peer DDN Mailbridge via another EGP session.

During mid-1989, the use of gateways on regional networks was replaced with a direct network. This was done quite simply by putting Ethernet adapters on the Mailbridge so they could use an Ethernet to connect the NSS to the Mailbridge. By November 1989, two of these Ethernet-based backbones were in place: one at NASA-Ames Research Center (Moffet Field, CA) and the other on the East Coast at Mitre (Reston, VA).

This higher-performance link allowed better distribution of loads. Instead of letting one gateway act as a hot link, the NSFnet decided to distribute the load to all operational gateways. This was accomplished by making a specific network number always prefer a particular Mailbridge. In case of outage, the other takes over the full load.

The NSFnet treats each of these DDN links as a different autonomous system. The routes learned via NASA are labeled AS 164, while the routes learned from Mitre are labeled AS 184.

NSFnet inbound routing is based on a distributed policy routing database. Each network number is verified against a list of legitimate networks. Each site associates a particular metric with that number.

For example, network 10 is the ARPANET. The NSS in Palo Alto learns about this network number and gives it a metric of 10. The College Park, Maryland, NSS also learns about network 10 and gives it a metric of 12. Currently, the DDN networks are randomly split between the two bridges. Eventually, they would like to know which is “closer” and assign each DDN network to the NSS closest to it.

When the NSFnet announces a metric to the DDN for a particular network, it does so based on the distance between the gateway NSS and the NSS that will interject the traffic in question into the backbone. This is basically a hop count.

For example, the Princeton NSS is one hop away from the NSS at Mitre but three hops away from NASA-Ames. Therefore, the Mitre Mailbridge will receive a better metric than the NASA-Ames Mailbridge. Thus, a DDN network wishing to send to Princeton would send to the Mitre Mailbridge, which goes to the Mitre NSS, which goes to the Princeton NSS and then on out.

Connectivity via the NSFnet to the DDN is arranged through a mid-level network, which will tell the NSFnet. Even if a network has a direct DDN connection, it may still want an NSFnet-based route as a backup. If so, the Mailbridges will get a very high metric from the NSFnet NSS.

Note that this is a somewhat simplistic environment. What if you want DDN-DDN communication to go via the NSFnet backbone? This might make sense since NSFnet is faster than DDN. That would be more of a general mesh structure, instead of what we saw in this RFC, which is a very limited point of connection.

## TCP on a Sun

The TCP module on a Sun is started up when the system boots. Typically, the `/etc/rc.boot` file has a series of commands in it to set things up. The `ifconfig` command configures parameters for each network interface on the system:

```
ifconfig ie0 hostname netmask + -trailers up
```

The `ifconfig` command in this case specifies that a particular interface (e.g., `ie0` for the first Ethernet card) has a certain host name. A network mask is specified as `+`, which means that a Network Information Service (NIS) map (i.e., `netmasks.byaddr`) is consulted.

The `-trailers` indicates that the interface will use a standard Ethernet format. Trailers is a method used in Berkeley Unix systems that puts the

header information at the end of the packet. Why do this? It means that when data comes in, it can be aligned on memory boundaries. If it is not aligned, it has to be copied into a buffer before presentation to the user. While trailers works in a strictly Berkeley environment, it causes havoc with general Ethernet traffic.

Finally, the `ifconfig` command puts the network interface up. In addition, a second command would be issued to set a broadcast address for the card and to control other parameters (e.g., specify that the interface is private).

Of course, if this is a diskless node, there is no need to initialize the Ethernet card. That process occurs before the diskless node boots—it needs the network to find the operating system.

In addition to commands in the `/etc/rc.boot`, there are a variety of configuration files on the system that store information (see Fig. 3-24). With many systems, these configuration files can become tough to manage. Chapter 9 shows how NIS can be used to replace many of these configuration files.

One of the key configuration files is the hosts database. It contains the mapping between IP addresses and host names (or nicknames or aliases). It allows the user to specify servers by host name instead of remembering the IP address (see Fig. 3-25).

Similarly, the `networks` file includes information on network names:

Network name	Network number	Nickname
Arpanet	10	arpa
Eng	193.9.0	
Corp	193.9.1	
Prog	193.9.2	

Once the configuration files are set up, there needs to be a provision to start up the appropriate daemons, which represent the different servers that will be available on UDP or TCP ports. Some of these daemons are programs that run all the time. For example, the RPC registration daemon is always running, waiting to respond to incoming RPC calls. Other daemons are programs that do not need to be memory resident. Instead of always running each of these services, a single daemon called the Internet Services Daemon (`inetd`) is started up. This daemon invokes services as they are needed.

The `inetd.conf` configuration file contains the various programs that might be used and specifies what port each uses. The daemon then listens in on these ports. Examples of typical programs served this way are FTP and Telnet.

A simple way to monitor the current status of a workstation on a TCP network is with the `netstat` command. Figures 3-26 and 3-27 show a variety



hosts	Host names and IP addresses.
ethers	Host names and their Ethernet numbers.
networks	Network names and numbers.
protocols	IP protocol names and numbers.
services	IP service names and their port numbers.
netmasks	Network names and their netmasks.

**3-24** /etc Configuration Files

```
# /etc/hosts file
# local net 1: 192.9.200: 10mbps Ethernet: Engineering
192.9.200.1    jekyll          # me
192.9.200.2    usher
192.9.200.3    raven
# local net 2: 192.9.201: Marketing
192.9.201.11   flugg
192.9.201.12   fluff
192.9.201.4    jeckyll-hyde   # me again
```

**3-25** An /etc/hosts File

of pieces of information for a Sun workstation on a TCP/IP network. Notice that information on routing tables and the operation of the protocol is available for each workstation. Using this information is up the individual network manager. Some managers would simply type netstat when things go wrong. In a larger environment, the information maintained by netstat would be accessed using SunNet Manager, discussed in Chapter 15.

In SunNet Manager, the network manager would signal certain events as being of interest. In Figure 3-26, for example, we might wish to be notified if the number of duplicate packets reaches a certain level. In the example, the number is 2 out of 5783, or roughly 0.04 percent of the total. If the number of duplicates reached 1 percent, we might wish to have an alarm sound.

Note that the cost of having some variable like this monitored might make the network manager more choosy about what to ask for. Every time some variable is monitored, a program must poll the system periodically, an activity that takes a small but nonzero amount of resources.



hydramatic% netstat -l

Name	Mtu	Net/Dest	Address	lpkts	lerrs	Opkts	Oerrs	Col
le0	1500	b14-spd-c	hydramatic	2729714	1	8479	1	208
lo0	1536	loopback	localhost	266	0	266	0	0

hydramatic% netstat -rs

routing:

- 0 bad routing redirects
- 0 dynamically created routes
- 0 new gateways due to redirects
- 0 destinations found unreachable
- 0 uses of a wildcard route

hydramatic% netstat -r

Routing tables

Destination	Gateway	Flags	Refcnt	Use	Interface
localhost	localhost	UH	1	92	le0
129.144.200.0	commserv	UG	0	0	le0
swan-cent	commserv	UG	0	0	le0
b5-spd-perf-ddlab	commserv	UG	0	0	le0
envtest-lab	commserv	UG	0	0	le0
b21-gpd-net-1	commserv	UG	0	0	le0
b10-springnet	commserv	UG	0	0	le0
b14-spd-fddlab-b	bodacious	UG	0	0	le0
analoglab	commserv	UG	0	0	le0
b17-dsd-hw-office-a	commserv	UG	0	0	le0
b8-gpd3	commserv	UG	0	0	le0
b1-labnet	commserv	UG	0	0	le0
b19-mps-sim	commserv	UG	0	0	le0
b16-wsg-c	commserv	UG	0	0	le0
b14-spd-a	bodacious	UG	4	396	le0
b14-ta-users	bodacious	UG	0	0	le0
fddi-net	commserv	UG	0	0	le0
b1-spd-dae-lab	commserv	UG	0	0	le0
b15-wsd-sst	rats	UG	0	0	le0
cte-net	commserv	UG	0	0	le0
b19-disknet	rats	UG	0	0	le0
b21-gpd-net-2	commserv	UG	0	0	le0
b18-ssd-bb	commserv	UG	0	0	le0
galaxy-net	commserv	UG	0	0	le0
b17-dsd-hw-office-b	commserv	UG	0	0	le0
b15-labnet	commserv	UG	0	0	le0
b8-gpd4	commserv	UG	0	0	le0
mtview	commserv	UG	0	1	le0
b19-visi-circuit	commserv	UG	0	0	le0
b16-wsg-d	commserv	UG	0	0	le0
b19-sundragon-hwl	commserv	UG	0	0	le0

hydramatic% netstat -s

udp:

- 0 incomplete headers
- 0 bad data length fields
- 0 bad checksums
- 0 socket overflows

tcp:

2169 packets sent

- 844 data packets (11571 bytes)
- 0 data packets (0 bytes) retransmitted
- 492 ack-only packets (415 delayed)
- 3 URG only packets
- 10 window probe packets
- 747 window update packets
- 73 control packets

5873 packets received

- 904 acks (for 11638 bytes)
- 25 duplicate acks
- 0 acks for unsent data
- 5135 packets (6052774 bytes) received in-sequence
- 2 completely duplicate packets (2920 bytes)
- 0 packets with some dup. data (0 bytes duped)
- 177 out-of-order packets (229988 bytes)
- 0 packets (0 bytes) of data after window
- 0 window probes
- 2 window update packets
- 1 packet received after close
- 0 discarded for bad checksums
- 0 discarded for bad header offset fields
- 0 discarded because packet too short

23 connection requests

19 connection accepts

38 connections established (including accepts)

44 connections closed (including 0 drops)

4 embryonic connections dropped

904 segments updated rtt (of 927 attempts)

12 retransmit timeouts

- 0 connections dropped by rexmit timeout

0 persist timeouts

3 keepalive timeouts

- 0 keepalive probes sent

```
3 connections dropped by keepalive
icmp:
  3 calls to icmp_error
  0 errors not generated 'cuz old message too short
  0 errors not generated 'cuz old message was icmp
  Output histogram:
    echo reply: 22
    destination unreachable: 3
  0 messages with bad code fields
  0 messages minimum length
  0 bad checksums
  0 messages with bad length
  Input histogram:
    destination unreachable: 17
    echo: 22
    #9: 1214
  22 message responses generated
ip:
  2446450 total packets received
  0 bad header checksums
  0 with size smaller than minimum
  0 with data size data length
  0 with header length data size
  0 with data length header length
  24 fragments received
  0 fragments dropped (dup or out of space)
  0 fragments dropped after timeout
  0 packets forwarded
  6 packets not forwardable
  0 redirects sent
  0 ip input queue drops
```

### 3-27 (Cont.) The Netstat Command

#### Starting a Connection

At this point, it is worth stepping back and seeing what happens when a user starts up a connection on the network. An interesting description of what happens when a TCP session is started up was printed in *ConneXions* by Greg Marshall. The information that follows is a condensation of that description.

Let's assume that our user wants to initiate a Telnet session to the chemistry department at the University of California at Berkeley:

```
myhost% telnet violet.cchem.berkeley.edu
```

We assume that our user has a priori knowledge of the name of the host desired. This information is typically stored on PostIt notes on the wall next to the terminal but could have come from an X.500 directory.

The Telnet program will use the Domain Name System to find the IP address. This information may already be present in a local name cache. If not, and we are outside the Edu domain, Telnet will start asking name servers. When we get to the root for Edu, it will deny knowledge of the address but will give us a pointer toward Berkeley.Edu. Eventually, the name resolver on the client gets an IP address (the name resolver probably being part of the Telnet command or linked to it). Telnet then needs to know a well-known socket number. In Berkeley Unix operating systems, we'd typically check `/etc/services`.

Telnet now asks the local TCP for a socket. When we have the socket, we try to set up a connection, passing in the destination address and TCP port. On the local side, our TCP gives us a port number. The other option is to use a stream to connect to the TCP module.

At this point, the two TCP modules need to set up a connection by exchanging synchronization bits. Note that when we do the SYN exchange, we need to make sure we pick an appropriate sequence number so we don't get fooled by old packets from a previous incarnation of this connection.

Once our SYN packet is put together, it is handed to IP, which checksums it and sends it to the other side. Note that this forces a transition in the other side from the closed, listening mode over to the SYN sent, listening mode.

The routing of the packet may go through an intermediate gateway. We check our subnet address mask and see if the address is local or not. If it is not local, we use our gateway (we may get back a redirect). At this point, we resolve the IP address to a local gateway (via ARP) and hand it on down to the network for transmission.

### For Further Reading

- David D. Clark, *Fault Isolation and Recovery*, RFC 816, July, 1982.
- , *IP Datagram Reassembly Algorithms*, RFC 815, July 1982.
- , *Modularity and Efficiency in Protocol Implementation*, RFC 817, July, 1982.
- , *A Subnetwork Addressing Scheme*, RFC 932, January, 1985.
- , *Window and Acknowledgment Strategy in TCP*, RFC 813, July, 1982.
- Comer, Douglas, *Internetworking with TCP/IP* (vol. 1, 2nd ed.). Prentice Hall (Englewood Cliffs, N.J.: 1991). The standard text on TCP/IP.
- C. Hedrick, *Routing Information Protocol*, RFC 1058, June, 1988.



Jacobson, V., *Compressing TCP/IP headers for low-speed serial links*. RFC 1144, February 1990, 43 pp. (Format: TXT=120959, PS=534729 bytes)

Jacobson, V., Braden, R.T., and Zhang, L., *TCP extension for high-speed paths*. RFC 1185, October 1990, 21 pp. (Format: TXT=49508 bytes).

D.L. Mills, *Exterior Gateway Protocol Formal Specification*, RFC 904, April, 1984.

Nicholson, Andy, et. al., "High Speed Networking at Cray" *Computer Communication Review*, p. 99, v. 21, no. 1, January 1991.

David C. Plummer, *An Ethernet Address Resolution Protocol*, RFC 826, November, 1982.

J. Postel, *Internet Control Message Protocol*, RFC 792, September, 1981.

———, *Internet Protocol*, September, 1981, RFC 791.

———, *Transmission Control Protocol*, RFC 793, September, 1981.

———, *User Datagram Protocol*, RFC 768, August, 1980

J. Rekhter, *EGP and Policy Based Routing in the New NSFnet Backbone*, RFC 1092, February, 1989.

Rose, Marshall, *The Open Book* Englewood Cliffs, N.J.: Prentice Hall, 1989. The best book available on OSI.

Marshall Rose, Dwight E. Cass, *ISO Transport Services on top of the TCP Version: 3*, RFC 1006, May, 1987

Sidhu, Gursharan S., et. al., *Inside AppleTalk*. Addison-Wesley (Reading, Mass: 1989).

Stevens, W. Richard, *Unix Network Programming*. Prentice Hall (Englewood Cliffs, N.J.: 1990). A look at TCP/IP and UUCP from the point of view of the Unix operating system. A valuable look at the internals.

X/Open, *X/Open Portability Guide, Networking Services*. Prentice Hall (Englewood Cliffs, NJ: 1988).

J. Yu and H-W. Braun, *Routing between the NSFnet and the DDN*, RFC 1133, November, 1989.



## CHAPTER 4

# STREAMS





# STREAMS

## Protocol Stacks and the Operating System

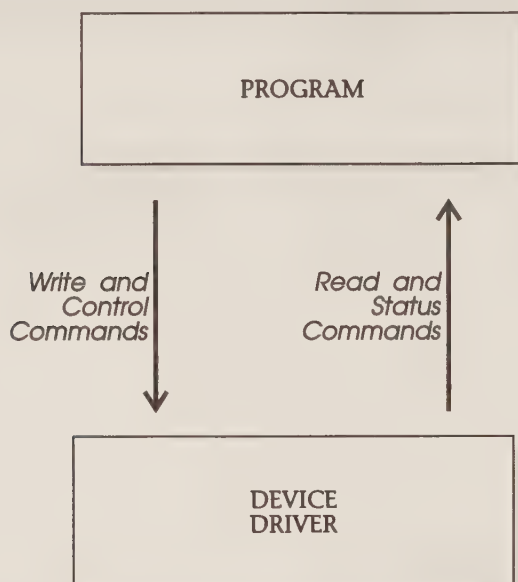
STREAMS is a service defined by AT&T as part of Unix System V Release 3. It was then incorporated into Unix System V Release 4, which merges several of the common mutations of Unix, including the Sun (BSD) and AT&T strains, into one common operating system.

STREAMS provides a way of connecting a series of modules together and passing messages down the chain. A typical example is building a protocol stack. Messages (protocol data units) are passed down the protocol stack and then out the device driver at the end of the stream. Incoming messages go from the driver upstream through each of the modules.

STREAMS provides the general structure for passing messages. The definition of a standard set of messages is part of an interface, such as the Transport Level Interface (TLI). TLI is a common interface defined by AT&T for user programs to communicate with a transport layer. An application that uses TLI is able to communicate with any transport layer that provides a TLI interface. Other interfaces also exist at other layers, such as the data link provider interface.

We will concentrate most on the Transport Level Interface because this is the technology used to provide a transport-independent RPC mechanism (TI-RPC). TI-RPC allows an RPC application to be designed in one environment such as TCP and run unchanged over other stacks such as UDP or TP4.

STREAMS and the other interfaces discussed in this chapter are not really networking tools in and of themselves—they are a standard interface on the operating system for accessing resources—both network-based and local. A standard interface does not provide interoperability by itself, but if used, it does make it easier for networking vendors to provide compatible equipment and software modules.



**4-1**  
Character I/O

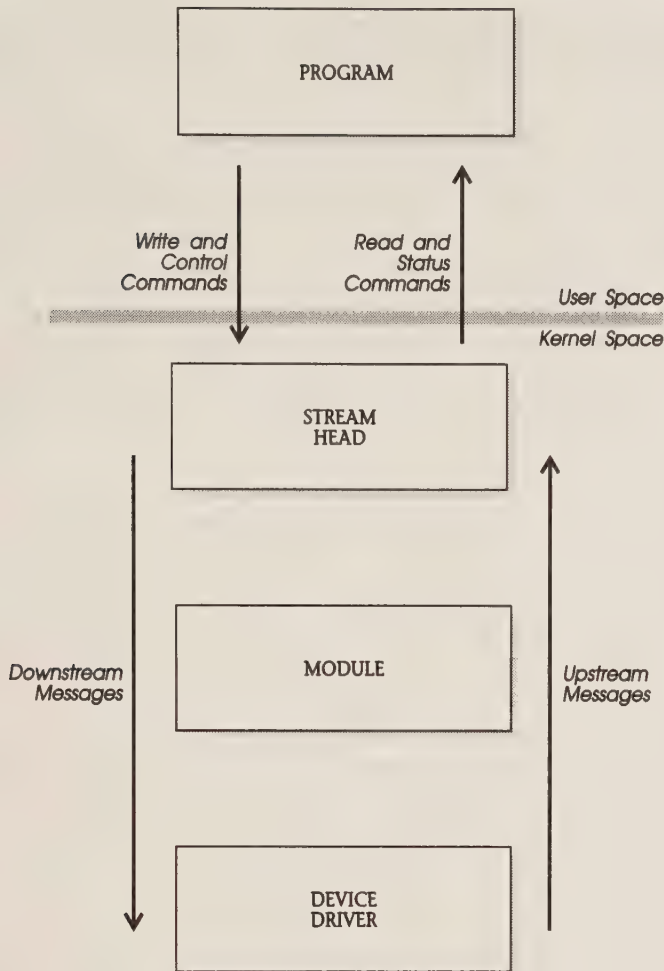
STREAMS is one mechanism for accessing network services. Prior to adoption of SVR4, the networking interface in the Sun Operating System was based on another abstraction, called Sockets. Sockets is simply a standard interface to the operating system, mapping a TCP or UDP port to an access point for an application.

Sockets has been implemented as a common application programming interface (API) for several different kinds of networks including XNS, TCP/IP, and internal Unix communications. Socket libraries exist on most Unix systems and are also part of PC-NFS. A more rudimentary service than STREAMS, Sockets is closely tied to the semantics of transport layer programming. In SVR4, a socket library is provided on top of STREAMS to provide backward compatibility for existing BSD applications.

### Character I/O Mechanisms

To understand STREAMS, it is best to start with the way input/output processing is done in a local environment. Any device to be accessed by a program, such as a modem, has a device driver. This device driver is able to accept characters and output them to the device—sending data through a modem (see Fig. 4-1). Each device driver has a set of commands that are used to signal the type of operation to perform. The idea behind STREAMS is to provide a common interface to all of these devices.

There is another significant enhancement in the STREAMS mechanism—support for message-based I/O mechanisms. Traditional device drivers are



**4-2**  
Basic STREAMS  
Mechanism

based on a character I/O mechanism. In STREAMS, the user can dispatch a message, which will be put into a queue for the device driver. The device driver then processes the message when it has time. STREAMS is message-based and can queue the messages. As will be seen, this provides an ideal framework for a networking environment, which also shares those characteristics.

### *Basic STREAMS Components*

A stream consists of three types of components: the stream head, modules, and a device driver (see Fig. 4-2). Modules are optional—all that is needed for a stream is the stream head and the driver. The stream head is what interacts with the program; it replaces the device driver as the interface the programmer sees.

A module sits in between the stream head and the driver. In the case of LAN, the driver might be an Ethernet driver. A module might be a software implementation of TCP, another IP, a third UDP.

The device driver is any STREAMS-compatible device driver. It is responsible for accepting messages from upstream—from the stream head or any modules. It takes that message, translates it, and sends it out to the device that it is driving. If the driver is an Ethernet driver, it sends data out onto the LAN. The driver also accepts data, which it sends upstream to the stream head, which in turn presents it to the user program.

As with a file, the stream head can perform basic functions such as reading and writing data. The stream head also supports an additional set of commands. When the stream is first opened, it consists of a driver and the stream head.

An application can push modules onto the stack between the stream head and the driver. The control functions of a stream allow the programmer to add and delete modules from the stream. To add a module to the stack, the programmer would use the `I_PUSH` argument to the `IOCTL` function call, giving the name of the desired module. To delete a module closest to the stream head, the `I_POP` option would be used instead. Additional modules are pushed between the stream head and the highest-level module. An application will send a command (e.g., an `IOCTL`) to the stream head, which takes the data and packages it as a message. The message is sent downstream. The first module downstream from the head examines the message and decides if it will take action. The module can modify the message, insert a new one, or even simply remove it from the stream.

If a module does not recognize a message, it sends it downstream to the next module. The next module downstream will examine the argument to the `IOCTL` and determine if it is a known type. We might define an `IOCTL` command for the transport module, for example, that was used to change the global retry limit for unacknowledged packets. Passing unrecognized messages downstream means that new functions can be added without breaking existing programs. When the message hits the end of the stream—the device driver—will discard any unrecognized messages.

The way that a program interacts with the stream head differs among operating systems. Usually, function calls are provided for control of the stream's operation, for support of traditional I/O mechanisms, and for getting messages from and putting them into the stream.

The programmer can issue a `putmsg` function call. That function call will alert the stream head that data from some location in memory is to be sent downstream. The stream head gets the data from the user's memory. The message has two parts: control and data. The data is what the interpreter of the message will use, the control part indicates the message type.

One scenario would be for a programmer to generate a series of Protocol messages. Protocol messages are interpreted by modules on the stream, modified, and then sent along to the next module or driver. For example,



the top level of the stream might be a transport layer that provides several service primitives, including establishing sessions and sending and receiving data.

This module would recognize a certain number of commands that could be in the command block of the protocol message. If the module sees an establish connection message, it copies in the data. Presumably, the data portion of this message would have the destination internetwork address of the desired remote host for this connection.

The transport module would then generate a message to be sent down to the network layer, which would add a routing header and send the message on down to the data link driver. In this way, each message is modified by the addition of a header and sent downstream through the protocol stack.

### *Message Queues and Flow Control*

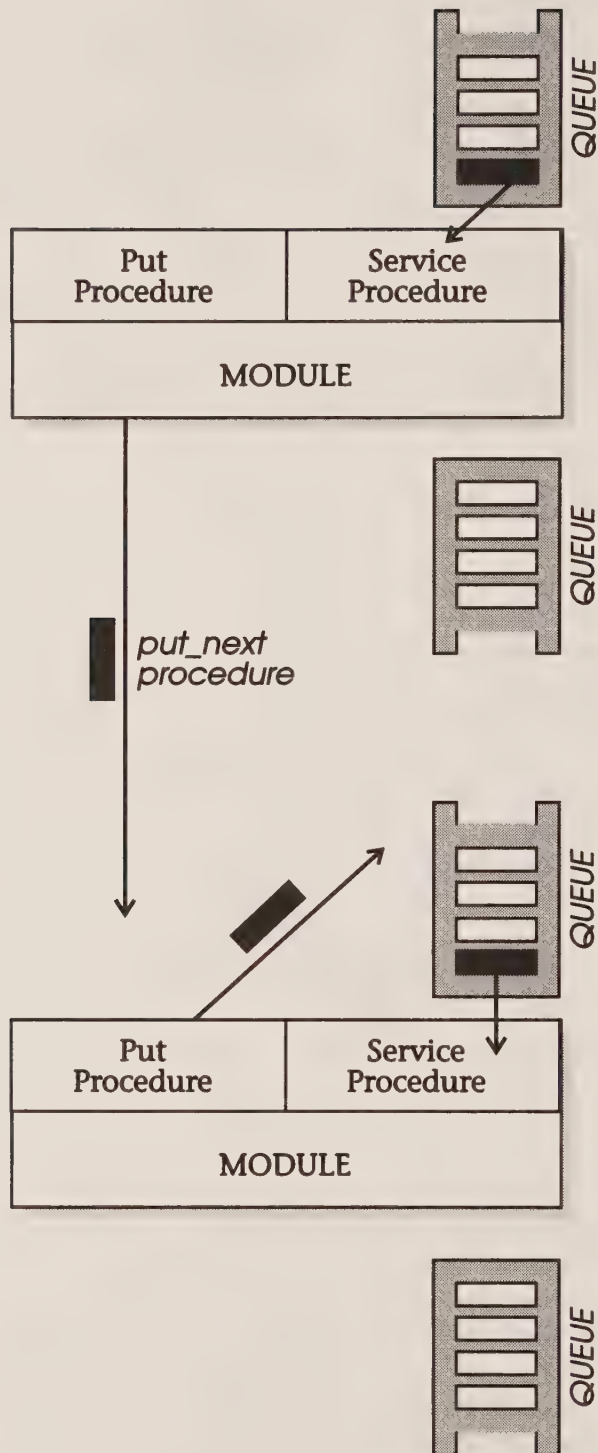
Each of the components on a stream has two message queues: one for incoming and one for outgoing messages (see Fig. 4-3). The queues provide flow control so messages do not overwhelm a module or driver, and they also provide expedited data services. A queue is simply an area of memory large enough to hold several messages. When a module wishes to send a message downstream, it invokes the `put_next` procedure, which submits the message into the read queue of the next module downstream.

The queue consists of a series of linked messages and ancillary control data. The head of the queue points to the first message, which in turn has the address of the next message. The last message, or the tail of the queue, has a pointer back to the queue head. When a module is finished with a message, it passes it downstream with a call to the `put_next` procedure. This will hand the message to the `put` procedure associated with the incoming queue of the next module. The `put` procedure examines the message and decides if it needs immediate servicing. If not, it can choose to put the message into the queue. Normally, the message is put in the tail of the queue, so the pointer is adjusted to point to the queue head.

A module can have a service procedure associated with it. Periodically, this service procedure is called, at which time it is responsible for trying to empty the queue of all messages.

There are times when a module will be unable to send a message downstream because the downstream module is unable to process it. When this occurs, the stream is blocked. If the downstream queue is blocked, the local module may not be able to empty its own queue, eventually leading to a stoppage all up the line until the downstream queue unblocks itself.

Flow control is based on the size of the queue. Each queue has high- and low-water marks associated with it. Upstream modules can add messages to the queue until it reaches the high-water mark. The upstream module is blocked after a queue reaches the high-water mark until the queue is emptied down to the low-water mark.



#### 4-3 Queue Management in STREAMS

Normally, messages are submitted at the end of a queue. Certain classes of messages are known as high-priority messages; they are not subject to flow control and are not added at the tail of the queue. Instead, they are added at the head of the queue, behind any prior high-priority messages.

## The STREAMS Multiplexer

A basic stream consists of one user, one driver, and optional modules. In most computing environments, it is necessary to share drivers among multiple users. STREAMS can be multiplexed, allowing several users to share a single stream. Multiplexed drivers are an essential component of important STREAMS applications such as those based on the Transport Level Interface.

A multiplexed stream is actually two or more streams stacked on each other (see Fig. 4-4). A multiplexing module is used to route data to the appropriate lower or upper stream. Note that the multiplexing can be in either direction: several upper streams sharing the multiplexing module or several lower streams under the control of the multiplexing module. Multiple lower streams are created by first opening two separate streams. The first consists of the stream head and the multiplexing module. The second stream consists of a stream head and a driver.

Next, the lower stream is connected to the multiplexing driver using an `I_LINK` command. This linking instructs the stream head of the lower module to send all upstream data to the multiplexing module. All data from the multiplexing module is sent to the lower stream head and then passed to the driver.

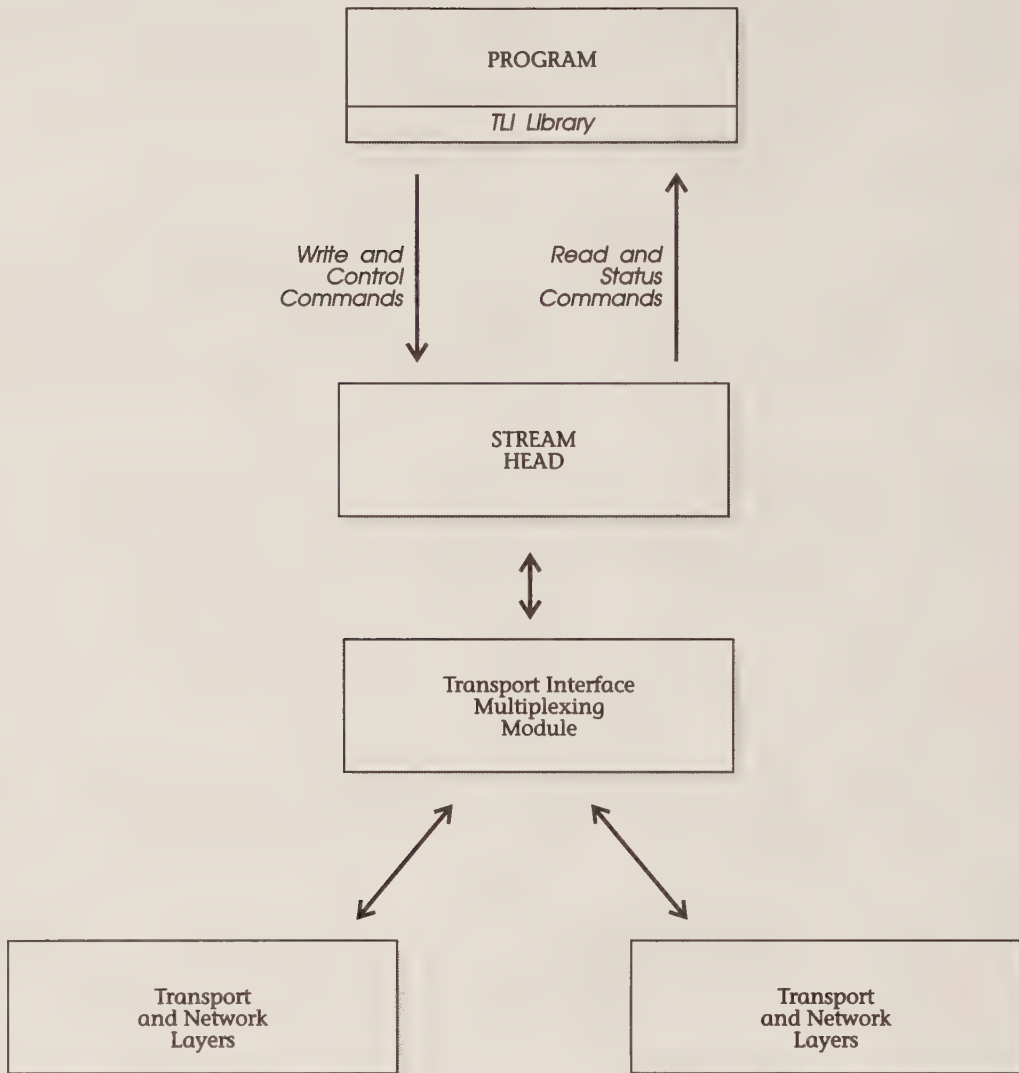
Additional lower streams can then be added using `I_LINK` commands. Several TCP modules could multiplex their services onto an IP module, which in turn would use the services of several different device drivers.

## Transport Level Interface

As we've seen, STREAMS provides a general mechanism for sending messages from one module to another and finally to the driver. If a module doesn't recognize a message, it passes it on. If the module does recognize the message, it will perform some processing and then usually send a revised message further downstream.

STREAMS provides a mechanism for passing these messages but doesn't define what those messages are. The Transport Provider Interface defines a common class of messages that are recognized by different transport layer service providers. The Transport Level Interface is the standard API used to generate TPI-compliant messages.

A module that is TPI compliant will provide service to any program that uses the TLI interface. Instead of programming directly to TCP, for example, the programmer would write an application that uses the TLI library



#### 4-4 Transport Level Interface

calls. At runtime, the program could be run using OSI and TCP/IP protocol stacks. Since the issue is transparent to the program, it is possible to change the underlying transport mechanism without rewriting programs.

TLI has another portability implication. Since many vendors have adopted the Transport Level Interface, at least the networking version of the program will be highly portable. If the programmer used a standard version of the C programming language, the rest of the code will also be portable.

TLI consists of two components: a TLI library and a TLI module. The TLI library is linked into the program and becomes a support library. The TLI



library accepts TLI calls that are in the main body of the program and performs the appropriate operations to deliver a message to the stream head.

The TLI module is opened when the stream is created. The module accepts requests from the TLI library and translates some of them into standard TPI messages. Below the TLI module are one or more different protocol stacks.

The TLI library allows the development of network-based programs without worry about what kind of transport stack is being used. For example, the programmer issues a `t_connect` function call to open a connection on any supported transport stack.

It is also possible to send out the connection request and not wait until it is established. The program would immediately continue with other processing. Then it would issue the `t_rcvconnect` call to find out if the connection request was successful.

A variety of TPI messages (and corresponding function calls) have been defined, including error handling and expedited data provisions. The basic functionality is the same as if the programmer was writing directly to the transport layer. The advantage of this approach is transparency—the programmer uses a single method to do network programming regardless of the type of network. Note that few users would be writing directly to the Transport Level Interface. Usually, the programmer would use a higher-level mechanism, such as the Netwise RPC Tool.

### For Further Reading

W. Richard Stevens, *Unix Network Programming*, Prentice Hall (Englewood Cliffs, N.J.: 1990).

Sun Microsystems, *STREAMS Programming Manual*, Part Number: 800-3826-10, Revision A of 27 March, 1990.



# RPC and XDR





## RPC and XDR

### Why an RPC?

At this point, the subnetwork, network, and transport layers of the network have been covered. Reaching the transport layer gives one the ability to send to and receive from some other node on the network. In many cases, through standard interfaces like TLI, the programmer does not have to worry about which transport provider (and hence which underlying network and subnetworks) is used.

Some applications, such as the File Transfer Protocol and most other Internet applications, take the transport layer as their interface to the network. The application opens a socket to a remote internet (and port) address and begins reading and writing data. For basic operations involving simple data transfer, this paradigm of the network is sufficient. However, there are a variety of other issues that can arise when two programs try to communicate across the network.

Take the representation of data in a heterogeneous network, for example. Integers and many other data types differ in their representation on different computer architectures. On one machine an integer might be stored with the high bit being the most significant and the low bit being the least significant. On another machine, the reverse convention might be used.

Another problem in defining a distributed application is finding an appropriate paradigm to communicate with remote portions of the application. Both the socket and TLI interfaces represent low-level network APIs. Neither provides a familiar model of programming for most application developers.

Since low-level interfaces do not address these issues, Sun added two more protocols called the Remote Procedure Call (RPC) and the External Data Representation (XDR). Remote procedure calls are a way that two programs can then issue a series of requests, replies, and error messages. Complementing RPC is the presentation layer protocol, XDR, which allows the

content of those messages to be interpreted in a heterogenous network. These protocols have formed a *de facto* standard for the upper portion of the protocol stack in a TCP/IP environment.

Built on top of RPC and XDR are services such as the Network Information Service (NIS) and the Network File System (NFS). NIS, discussed in Chapter 9, is a way to find services and other information on a network. NFS, discussed in the next chapter, extends the concept of a file system to the network.

This collection of protocols is marketed under the name of Open Network Computing (ONC), but NFS is often used as a shorthand way to refer to the whole family of protocols. ONC is sold by over 90 vendors, and implementations are available for all the major operating systems, including PC-DOS, VAX/VMS, all Unix mutations, VM/CMS, Macintosh, and MVS.

The question of RPC/XDR will be addressed in two phases. In this chapter, the protocol used on the network is examined. This protocol is usually accessed through an RPC/XDR library that is linked into application programs. The programmer simply places calls to the library to make calls across the network.

Direct programming of RPC/XDR was the first method available and many large applications, particularly NFS, have been written with this method. Using the RPC library shields many of the issues in network access from the programmer, but it still requires some knowledge of what is happening in the network.

A higher level of abstraction is using an RPC compiler. This is a higher-level language that allows the programmer to treat a collection of computers as a single system. The programmer designs procedures with inputs and outputs and then calls those procedures much the same as local procedures.

RPC compilers are discussed in Chapter 7, after we look at a sample application. The break in continuity was done for two reasons. First, looking at the basic RPC application gives the reader an appreciation for the power of this paradigm. Before we move on to how distributed programs are written, it makes sense to look at why one would want to write one. Second, we will see that we achieve a great deal of value by adding a good RPC compiler to the basic protocol. The basic protocol is a way of moving messages over the network. Conveniences, like sophisticated error handling or debugging, are not really built into the protocol. These mechanisms are often provided, however, using the additional layer of code provided with an RPC compiler.

We are thus looking at two levels of RPC programming. First, there is the information that actually ends up on the network, the protocol. Second, there is the interpretation of those calls by a network library before passing the calls on to the appropriate server or client program.

## *The Protocol*

RPC is a very simple protocol. All messages are either a request or a reply. Each message is identified by a 32-bit transmission ID, used to correlate replies to requests.

There are also three 32-bit numbers to specify which task is required in a request: the program number, program version, and procedure number. Some server programs support multiple versions; others will only accept a single version of the command. Version numbers allow graceful upgrades to services.

An RPC call can have several arguments and several return values. However, some compilers limit the number of return values—early versions of `rpcgen`, for example, would only provide a single result (although that result could be a complex data structure). The RPC Tool accepts procedures with multiple arguments and return values. Each of the arguments might be a complex structure. By convention, procedure zero takes a void argument and returns a void result, allowing a program to be “pinged.”

The program number is allocated to an interface: a series of related procedures, all available by contacting the same server program. In one sense, the set of procedures in a program provide a network library. It is possible that a single server application may provide services via several different program numbers. Program numbers are allocated within groups. Group 1 is Sun administered and includes all the commercial programs. Group 2 is only for local use. Group 3 is transient, used for callback RPC. Finally, group 4 is reserved for future use.

Notice that we are identifying an RPC service by a program number, not by some well-known TCP or UDP port. A special service, `rpcbind` (also known as the portmapper), is used to notify a remote user which address a particular service is using at the time. This flexibility means that there can be many different RPC services, none of them tying up well-known port numbers for occasional use.

There is a minor bootstrap problem: how do we find the portmapper on the remote server to find out which port a particular application is using? To solve this, the portmapper is the only well-known RPC service: it uses a fixed address.

## *Accessing the Library*

Figure 5-1 shows the basic RPC calls that are available to the programmer. The calls are split into a few categories, the first two being calls available to client or server programs. After that are some miscellaneous calls like Authentication, an issue addressed in more detail in Chapter 10.

Figure 5-2 shows a typical RPC call across the network. The RPC packet (embedded inside a UDP or TCP header) contains a transaction ID, so that



RPC Server Calls
Register procedure with RPC service package.
Wait for RP requests to arrive and call appropriate service.
Generic transport-independent RPC service creation.
Create RPC service transport for testing.
Create RPC service using TCP transport.
Create RPC service based on UDP transport.
Associate program number and version number with a service dispatch procedure.
Decode arguments of an RPC request.
Get the network address of the caller of a procedure.
Send back results of an RPC.
Free data allocated by RPC/XDR when decoding arguments.
Refuse service because of failed authentication.
Called when service cannot decode parameters.
Service has not implemented this procedure.
Program not registered with RPC service.
Version is not registered with RPC service.
Service detects system error.
Insufficient authentication.
Remove mapping of program and version to a dispatch routine.
Destroy RPC service transport handle.
RPC Client
Call remote procedure.
Broadcast remote procedure.
Create generic transport-independent RPC client.
Create RPC client for testing.
Create RPC client using TCP transport.
Create RPC client using UDP transport.
Global variable indicating reason client create failed.
Print message to stderr about why client handle creation failed.
Call remote procedure associated with client handle.
Copy error information from client handle to error structure.
Print message to stderr corresponding to error number.
Print message to stderr about why clnt_call fails.
Free data allocated by RPC/XDR when decoding results.
Destroy client's RPC handle.



<b>Authentication</b>
Return RPC authentication handle with no checking.
Return RPC authentication handle with Unix permissions.
Return default Unix authentication handle.
Return RPC authentication for Secure RPC.
Convert operating system-independent netname into locally usable credential.
Destroy authentication information handle.
<b>Secure RPC</b>
Convert host or domain-specific host name to operating system netname.
Get netname for a specific user.
Convert netname to host-specific name.
Convert operating system-independent netname to domain specific user ID.
Convert user ID to netname.
<b>RPCBind</b>
Establish mapping between program and its address.
Return list of RPC program to port mappings.
Return port number on which service is waiting.
Instruct portmapper to make an RPC call.
Destroy mapping to port.

5-1 (Cont.)  
RPC Library Calls

requests, replies, and errors can be matched. The program being called in this case is the Network File System. Procedure number 4 is called, which is used to look up a file name.

Note that the purpose of the call is immaterial to the RPC modules. All RPC does is to pass on calls to the server program, in this case the NFS daemon running on the server machine.

As Chapter 10 will show in more detail, the credentials allow a server program to determine who is using the program. Once the client is authenticated, the server can decide what sort of access policy to provide. Authentication in RPC is provided by different “flavors.” In Figure 5-2, the flavor is Unix, meaning that a UID for the client is passed over to identify the user. Users can add additional flavors.

Figure 5-3 shows a transaction being sent, this time to another program. The authorization flavor in this case is 3, which happens to be the Secure RPC flavor, based on public key cryptography.

For each flavor that RPC knows about it will examine the credentials and tell the server program if the credentials are well formed. Different flavors

provide different kinds of information. The Unix flavor gives the UID of the user, whereas Secure RPC gives a network-wide username.

How to interpret this information and the policies to set about what services should be provided to the remote user are not the province of the RPC mechanism. RPC simply delivers the information to the remote site and then accepts an answer (see Fig. 5-4).

## The Portmapper

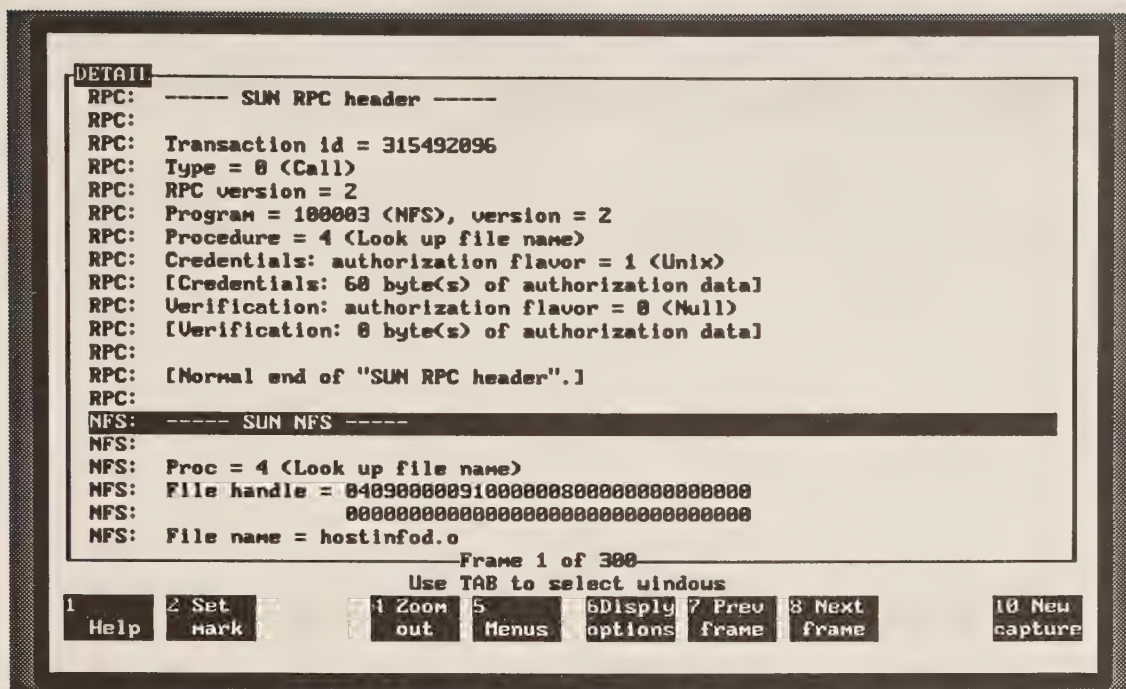
To find the current address for a program on a server, the portmapper or `rpcbind` service is used. The portmapper is an RPC program that has been assigned program number 100,000 (see Fig. 5-5). This RPC program is the only one that listens on well-known TCP and UDP ports. Any system that is set up to accept RPC calls will have at least this one application running. Figure 5-5 shows that a client program has sent a request to the portmapper to look up the current status of program 80955. Figure 5-6 shows that the program is currently accepting calls on port 1073.

At this point, most programs would then set up a TCP connection (or send UDP datagrams) to the destination program. We have assumed that the client program has a priori knowledge of the existence of the program on this particular node. One way of finding the current location of a service is to use another program, the NIS name server. A typical client initialization would request the name of a host system that provides a service. Then the portmapper on the host system would be contacted to find the current address of the service. Next, the client would request, through the RPC client library, that TLI establish a connection to a particular combination of port and network address.

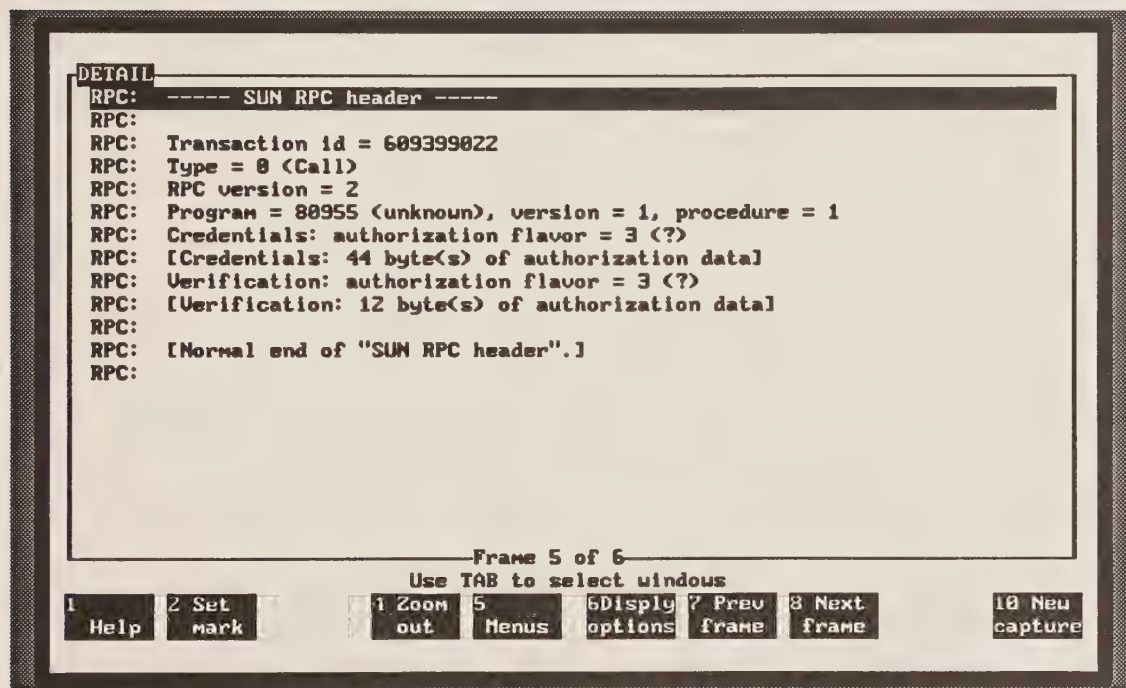
Many of the well-known RPC services are represented by daemons on server systems that start up at boot time. There are some programs which provide valuable services but are infrequently called. For these programs, a special daemon called the internet daemon (`inetd`) can be used as a proxy that will listen on the transport port for incoming connection requests and will invoke the service when the port receives a message.

Figure 5-7 shows two examples of the `rpcinfo` command, which uses the portmapper service to find out which programs are ready and waiting for requests. The first command asks for information from server `commserv` and the “mount” service, used to start up NFS sessions.

The reply from `commserv` is that there are two versions of the mount program available on the server. The `rpcinfo` command first consulted NIS to find out the network address of the host “commserv.” Then, portmapper requests were sent to that host to find out the status of a particular program.

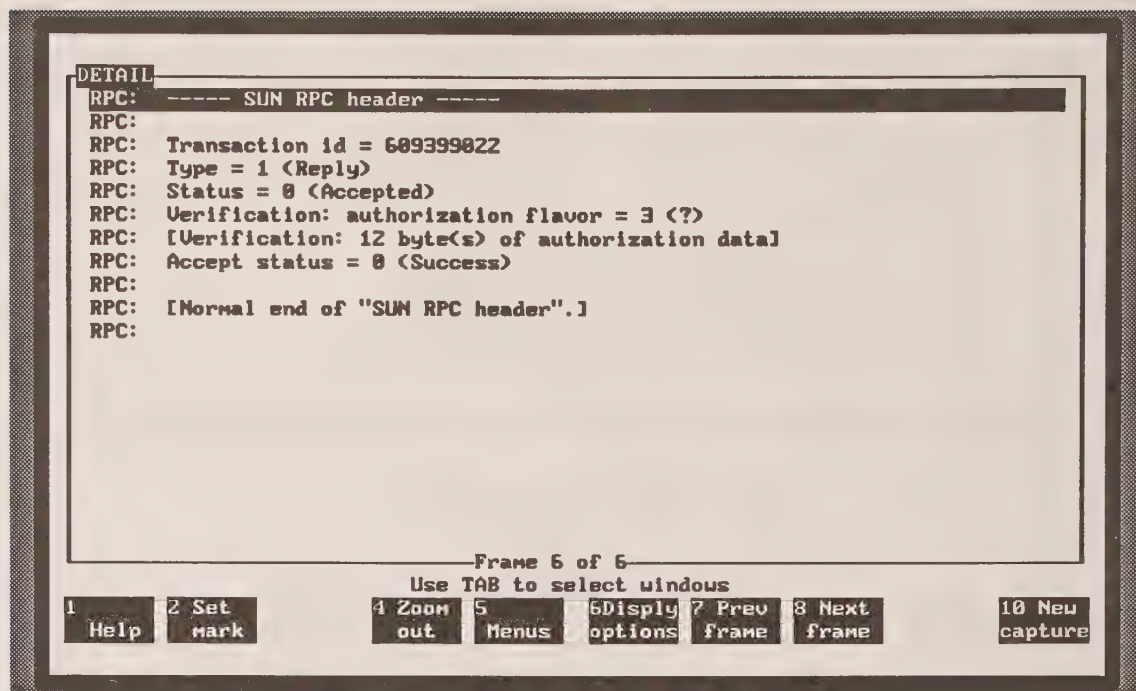


5-2 An RPC Header

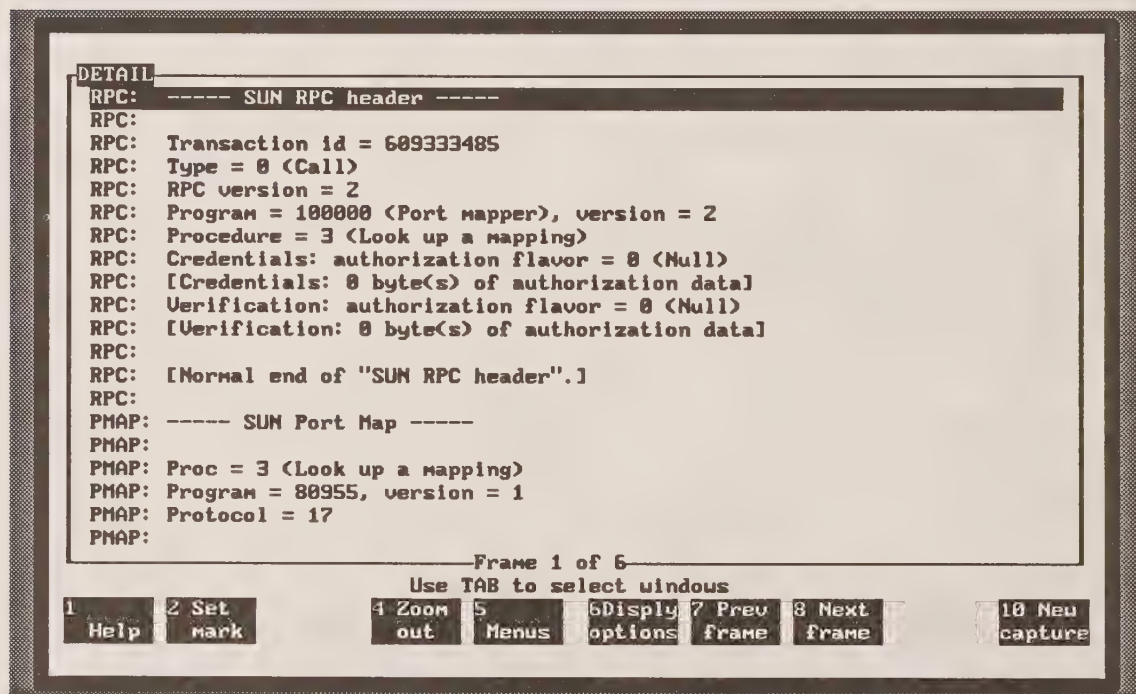


5-3 Adding an Authorization Flavor



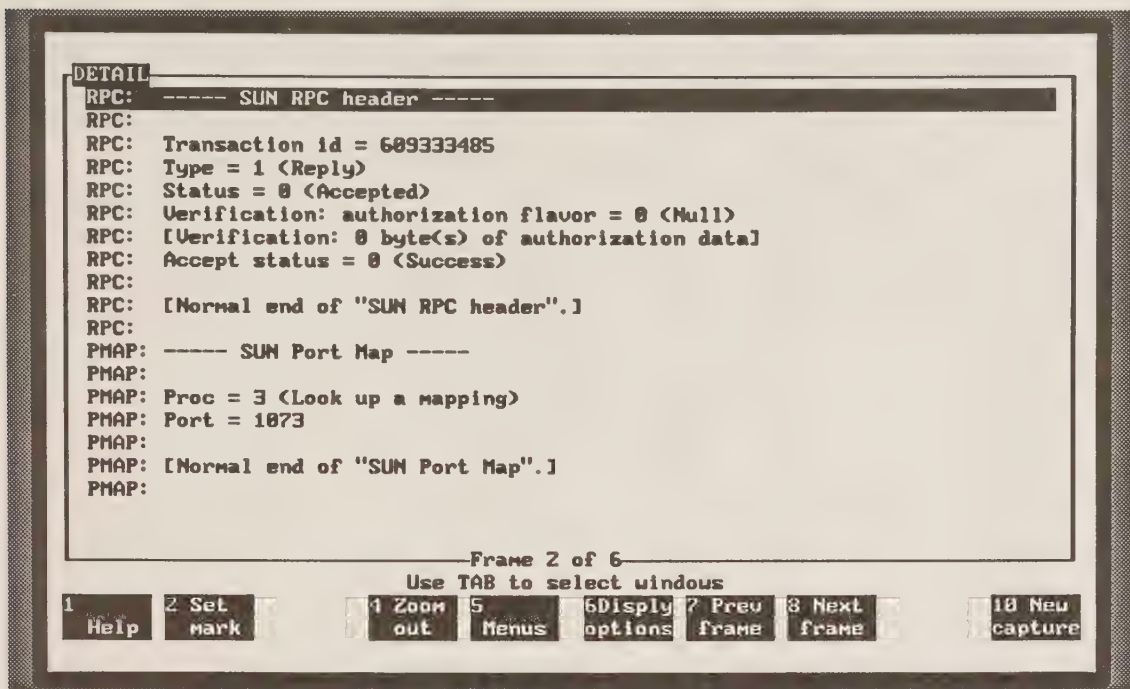


5-4 Authorization Accepted



5-5 The Portmapper





5-6 The Portmapper Reply

It is possible to find out all RPC-based programs that are currently available on a system, as shown in the second command in Figure 5-7. Many of these programs will be examined in subsequent chapters. Ypbind, for example, is a program used by clients looking for name servers in NIS. Key-serv is used for public key cryptography in Secure RPC.

### *Transport Selection Issues*

The selection of a transport mechanism is usually left up to lower layers of the network. The programmer may simply not care. There are some instances where it might make a difference.

For example, TCP is a reliable protocol, whereas UDP is not. RPC will adapt its retry policy accordingly by being more aggressive in a UDP environment, but it is possible (though not likely) with UDP that a remote server may never get the request. If the server never gets the request, then the client certainly will not get a reply, so the RPC code will time out and return an error message. Nevertheless, the programmer may wish to ensure what is known as at-most-once semantics: delivery of the request once or not at all, but not multiple times. TCP is an example of a transport protocol that provides at-most-once semantics.

One issue that is often brought up to justify the use of UDP over TCP is the protocol overhead. For a single packet, there is no doubt that UDP has

```
hydramatic% rpcinfo -u commserv mount
program 100005 version 1 ready and waiting
program 100005 version 2 ready and waiting
```

```
hydramatic% rpcinfo -p commserv
program vers  proto  port
100000    2      tcp    111   portmapper
100000    2      udp    111   portmapper
100007    2      tcp    1024  ypbind
100007    2      udp    1027  ypbind
100007    1      tcp    1024  ypbind
100007    1      udp    1027  ypbind
100029    1      udp    665   keyerv
100028    1      tcp    670   ypupdated
100028    1      udp    672   ypupdated
100003    2      udp    2049  nfs
100005    1      udp    717   mountd
100005    2      udp    717   mountd
100005    1      tcp    720   mountd
100005    2      tcp    720   mountd
100026    1      udp    725   bootparam
100024    1      udp    729   status
100024    1      tcp    731   status
100021    1      tcp    732   nlockmgr
100021    1      udp    1037  nlockmgr
100021    3      tcp    737   nlockmgr
100021    3      udp    1038  nlockmgr
100020    1      udp    1039  llockmgr
100020    1      tcp    742   llockmgr
100021    2      tcp    745   nlockmgr
100021    2      udp    1040  nlockmgr
100011    1      udp    1042  rquotad
100001    2      udp    1043  rstatd
100001    3      udp    1043  rstatd
100001    4      udp    1043  rstatd
100012    1      udp    1045  sprayd
100008    1      udp    1046  walld
100109    10     udp    1047  activity
100118    10     udp    1048  etherif
100117    10     udp    1050  hostif
100112    10     udp    1051  hostmem
```

less overhead than TCP. However, careful coding strategies in TCP implementations and a common scenario of multiple packet exchanges make the TCP overhead fairly minor.

TCP tends to have the advantage when very long arguments have to be sent. UDP is limited to datagrams of 8 kbytes, so services that need very long arguments use TCP. On the other hand, when an error occurs in TCP, the user program is forced to take actions to reset the connection. With UDP, we simply try again until we succeed (unless the server has gone away and the client must reset the binding). The Network File System is an example of an RPC application that uses UDP.

Finally, UDP is considered lightweight. For services accessed by multiple clients, connection management may be troublesome. UDP is simple to use for servers supporting many clients.

With transport-independent RPC, the user is even further removed from the question of selecting a transport mechanism. Typically, the transport mechanisms are in one of three classes: connectionless (datagram), connection-oriented with abrupt release (e.g., TCP), or connection-oriented with orderly release. The program can specify which one it prefers or leave the decision up to the user or a default. In the `/etc/netconfig` file there will be a list of available transports. Each user or program can set preferences with the `$NETPATH` variable.

In the TI-RPC arena, an interesting question is how to do address resolution. After all, if the transport has not been determined, the address can't possibly be known. The successor to the portmapper, `rpcbind`, handles this particular problem. Services register themselves with `rpcbind`.

### *Broadcast RPC*

Broadcast RPC allows a single call to be sent to multiple servers in a subnetwork. Broadcast RPC is used for applications such as data collection or for finding a location-independent server such as the nearest NIS name server. If a request is broadcast to every server at the same time, it can be assumed that the one that answers first is the one with the most available resources—a primitive form of load balancing.

The basic issue in constructing a broadcast RPC call is how to wait for replies and how many replies to wait for. If you are only interested in the fastest server, you only need a single reply.

To use a broadcast RPC mechanism, there is no need to change the server application. Just because a particular client is working with multiple servers is of no concern to any one server. The client has to figure out how to maintain multiple sessions. It is, of course, possible to build semantics into the server applications that make them aware of the presence of multiple servers, but the protocol doesn't require this.



Broadcasting is certainly an instance in which the transport mechanism matters. For example, TCP does not offer a broadcast service, so UDP must be used instead. Since UDP is limited in size, it is important to make sure that the broadcast packet fits not only into a single UDP packet, but also into the maximum transmission unit (MTU) of the underlying networks. On an Ethernet, the MTU would be about 1500 bytes.

One more issue needs to be addressed if IP is the underlying network protocol. In the IP protocols, a broadcast is available but must go to the same IP port address on each machine on the network. The only service that fits that criterion is the portmapper. All broadcasts are thus addressed to the portmapper, which is responsible for sending the request on to the appropriate server if one exists on that machine. If the server doesn't exist, the portmapper ignores the broadcast request.

An example of broadcast RPC is the `rusers` command with no arguments. `Rusers` tells who the users are on a machine. If the `ruser` command gets no arguments, it sends a broadcast.

Broadcasts of any sort, including RPC broadcasts, can easily cause all sorts of congestion problems if spread over a wide area. For this reason, IP broadcasts are never forwarded over gateways and are limited to a single local network (or possibly even just a subnetwork).

## Asynchronous Mechanisms

Normally, the RPC mechanism is synchronous: a request is sent and the calling process blocks until a reply is received. There are two methods used to change this basic behavior:

- Nonblocking RPC
- Callback RPC

Nonblocking is a one-way message facility: no path is provided for replies. Typically, after a string of nonblocking calls are sent a normal blocking or callback RPC call would be placed to get a response from the server. A nonblocking request is simply one where the timeout is set to zero and timeout errors are ignored. The client code will get back void results, and the server procedure simply doesn't answer the request.

Note that if nonblocking requests are used, the RPC library can no longer assure that things have worked properly. A request sent over UDP could be lost and the sender wouldn't know about it. If TCP is used, at-most-once semantics are guaranteed.

Callback RPC can be used with a nonblocking request or with a normal request/reply synchronous call. Callback RPC uses the transient address space by having the client and the server switch roles, allowing more of a



peer-to-peer paradigm. A single client can receive callbacks from multiple servers.

A typical example of a callback RPC mechanism would be a mail system. When a mail system receives new messages, it needs to contact the user agents. Normally, the user agent is the client and the mail system is the server.

With callback RPC in the hypothetical application, the server would start out as a client: it would place a call to the user agent, letting it know that mail has arrived. Later, when the user agent had some time, it would place a call as a client, using the address provided, to collect its mail.

The way this works is that the client creates a pseudo-service using a transient program number. It is selected at random out of the transient range and registered with the local portmapper to ensure that it is unique. Then, that information is sent to the server as part of the RPC request.

Note that at this point the server typically replies to the original request with a simple acknowledgment. A period of time then elapses while the server does some work. When the request is processed, the server will initiate a callback to the supplied transient number. The intermediate acknowledgment can be cut off by having the initial request be nonblocking with a timeout of 0.

In order to handle the callback, the client needs to set itself up as a server. It does so by using a call to the service run system call.

### *Batch Mode RPC*

Batch mode RPC is a way to send a whole series of calls at once before getting a reply. A series of nonblocking RPC calls are sent. Then, a request is sent with a nonzero timeout, which flushes all existing calls to the server. Presumably, the server application has been trained to not reply until it gets the last call. TCP is recommended as a transport layer so packets are not lost.

### The XDR Library Package

The XDR library package contains a set of routines that convert basic C language data structures into a serial stream of XDR encoded structures and vice versa (see Fig. 5-8). The same routine does both encoding and decoding (plus memory allocation for variable-length structures like strings).

Each of the C routines handles a translation for a set of primitive types. In addition, they can be combined to do things like linked lists or structured records. There is also support for pointers (specific support for pointers is needed since a pointer value is a machine address and won't mean anything on another machine).

Routine	Description
<b>Data Type Primitives</b>	
xdr_array()	Translate arrays.
xdr_bool()	Translate booleans.
xdr_bytes()	Translate counted byte strings.
xdr_char()	Translate characters.
xdr_double()	Translate double precision numbers.
xdr_enum()	Translate enumerations.
xdr_float()	Translate floating point number.
xdr_int()	Translate integer.
xdr_long	Translate long integer.
xdr_opaque()	Translate fixed-size opaque data.
xdr_reference()	Chase pointers within structures.
xdr_short()	Translate short integers.
xdr_string()	Translate null-terminated strings.
xdr_u_char()	Translate unsigned characters.
xdr_u_int()	Translate unsigned integers.
xdr_u_long()	Translate unsigned long integers.
xdr_u_short()	Translate unsigned short integers.
xdr_union()	Translate discriminated unions.
xdr_void()	Always return true.
xdr_wrapstring()	Package strings for calling from RPC routine.
<b>Stream Creation</b>	
xdrmem_create	Initialize an XDR memory stream.
xdrrec_create()	Initialize an XDR stream with record boundaries for TCP transport.
xdrstdio_create()	Initialize an XDR standard I/O file stream.
xdr_destroy()	Destroy stream and free associated memory.
<b>Miscellaneous</b>	
xdr_inline()	Invoke the in-line routines associated with XDR stream.
xdr_getpos()	Return current position in XDR stream.
xdr_setpos()	Change current position in XDR stream.
xdrrec_endofrecord()	Mark XDR record stream with an end of record mark.
xdrrec_eof()	Mark XDR record stream with an end of file mark.
xdrrec_skiprecord()	Skip remaining record in XDR record stream.
Source: Sun Microsystems	

## 5-8 XDR Library Calls

The primitive data types in XDR (see Fig. 5-9) can be combined to form various complex data types. Note that for all the primitive data types, data is represented in 32-bit units: this means that for an 8-bit character, the full 32 bits are used with 24 bits of zero. For character strings, data is packed together and padding added to reach the next 32-bit boundary.

Signed Integer	8, 16, 32, 64 bits
Unsigned Integer	16, 32, 64 bits
IEEE Floating Point	32 and 64 bit
Boolean	
Enumeration	
Fixed- or Variable-Length Arrays	Array of arbitrary type, including an array of structures
Fixed- or Variable-Length Byte String	
structures	
Discriminated Unions	
Opaque Data (Cookies)	

### 5-9 XDR Primitive Data Types

The actual data that the routines use comes from an XDR stream. The XDR stream can be from several different kinds of sources:

- Memory
- TCP/IP with record markers
- Unix standard I/O (which includes files and devices)

TCP provides a stream-like interface to the XDR library, which causes a problem: the library doesn't know when one structure is finished and another is starting. XDR puts in record boundaries which are then used by the remote XDR. XDR also supports standard Unix I/O so that not only files but any other devices that can support basic read and write calls are supported.

XDR takes all data going out onto the network and puts it into the standard XDR format. XDR defines a standard (canonical) network representation for data. All clients and servers translate data into and out of this standard format. In the case of two computers which use the Sun architectures (i.e., Motorola 68000 and SPARC) the XDR architecture happens to be close to the internal representation. Of course, this means that less work has to be done.

The worst case for translation is when a VAX is using XDR to talk to another VAX. Since the VAX is quite different from the Sun, it translates data, requiring a bit of effort, into XDR format. At the remote VAX, the data is translated again.

#### *Using XDR*

XDR can be used in conjunction with RPC or by itself. The application defines the encoding of information. For example, data can be simply en-



coded as a long byte string. This is the approach used by NFS when it makes remote files available to client programs.

This is appropriate for a service like NFS which tries to make remote files appear like local files. NFS does nothing with the contents of the data. It is up to a program to interpret what the data is supposed to mean. One file might be an operating system for a diskless node, another a word processing program executed by the operating system, a third a graphics file to be interpreted by a PostScript-speaking device.

Graphics files, as well as other data files, are an example of a case when XDR might be used to provide more structure to the information. Take the example of a central data source with a set of files. Located throughout the network are a series of application programs, each running on a different machine architecture, but all accessing the same data. One application program, say on a VAX, might write some data to a file. Another application program, say on a Sun, would then try to read the data file. NFS makes the data available without any knowledge of RPC or the network. The file appears local to both the Sun and the VAX. The VAX would write integers one way and the Sun would read them another way.

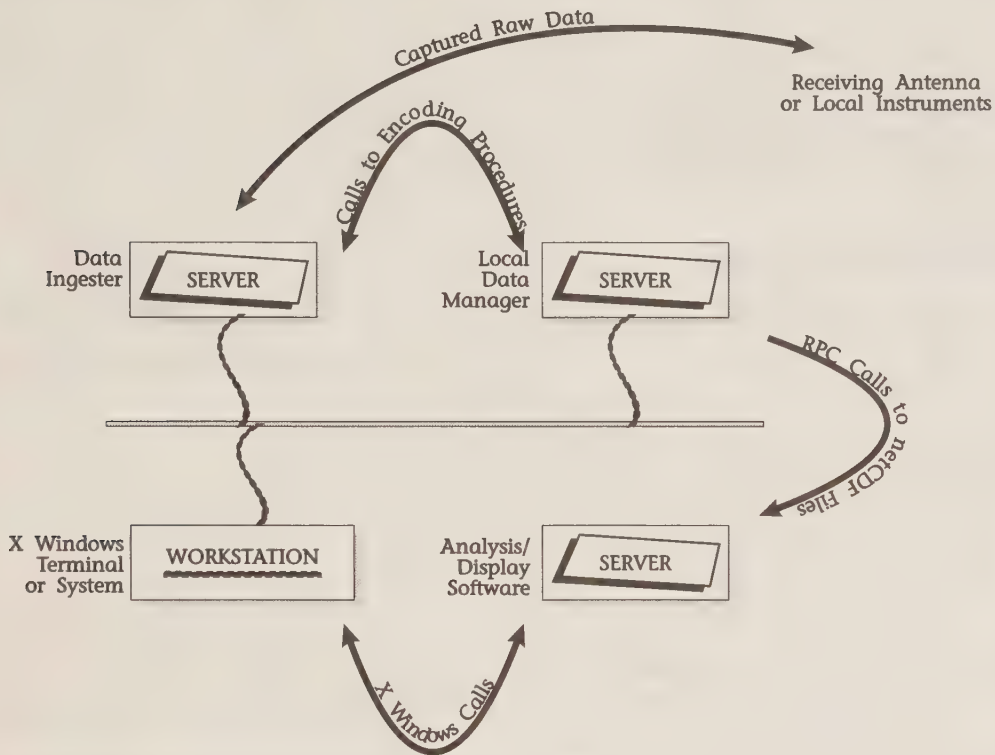
### *An Example Use of XDR*

A simple solution to this problem is to have both the Sun and the VAX set up XDR streams to the files instead of writing directly. In this way, the data is stored in a machine-independent format. This is the approach taken by Unidata in a program they wrote for the University Corporation for Atmospheric Research. Unidata was charged with developing an architecture to support the cheap distribution of weather data to universities and other facilities. The basic architecture is based on RPC and XDR, as shown in Figure 5-10.

Data comes into a campus by satellite or through local receiving instruments, where it is received by a computer known as the data ingester. The data ingester takes this real-time stream of data and encodes it in netCDF for storage on a file server. NetCDF builds one more level on top of XDR. XDR provides a way for data to be represented in a machine-independent format, but it provides no way for the remote node to know the exact structure. In most RPC applications, this knowledge of the structure of the incoming data is built into the XDR pack and unpack procedures. This assumes that we have defined this particular stream of data already and built that into an XDR routine.

CDF is NASA's Common Data Format, which provides a way for data to be self describing. NetCDF took NASA's way of describing what the data is and combined it with XDR for representation of the data (with a few more enhancements added in). Since the data is stored in this netCDF format, once it is at rest on the local data manager it is self describing. At this





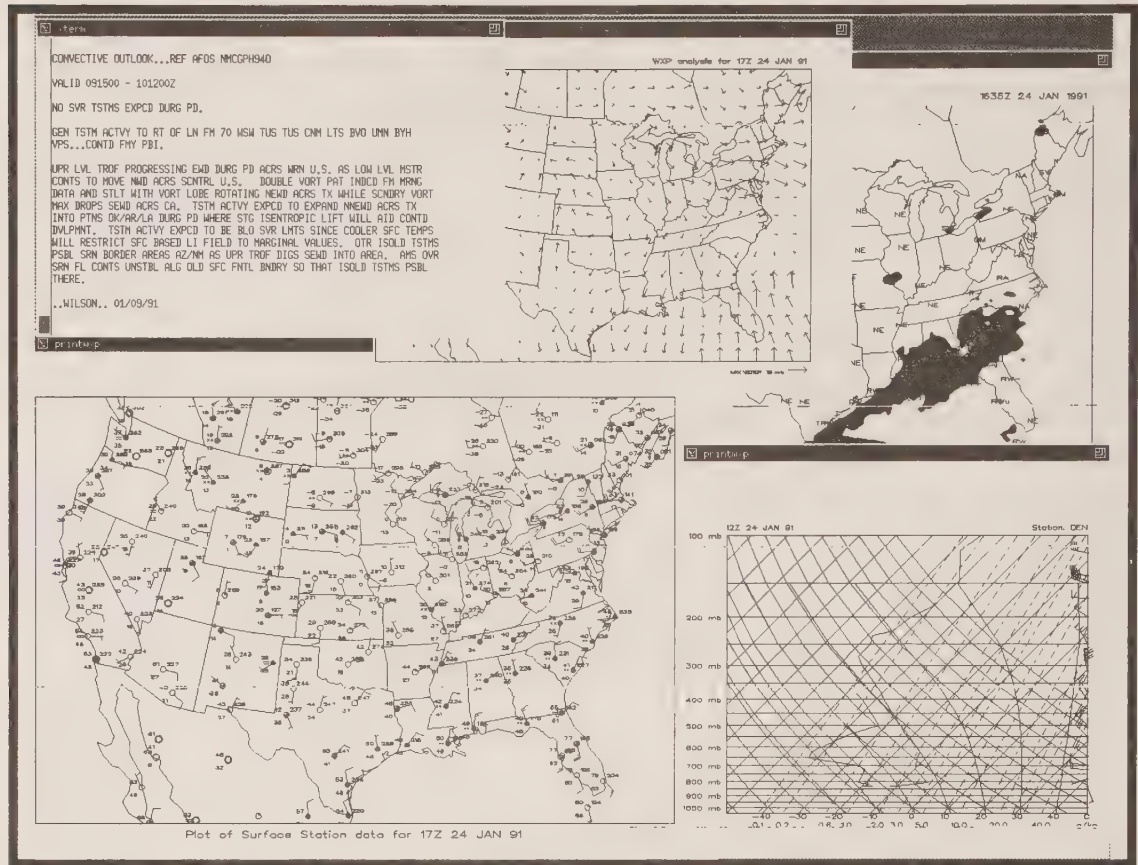
Data files are tagged using netCDF so they are self-describing, and then encoded in XDR so they are machine independent.

### 5-10 The netCDF Data Collection System

point, various analysis programs can place calls for data. The netCDF library allows easy access to very large amounts of 2D and 3D data.

There are different application programs that can make use of this information. This analysis/display software can come from a variety of sources. For example, Figure 5-11 shows an example of a screen from Purdue's University Weather Process (WXP) software. This software places calls to the local data manager to gather information.

The actual display of the data can well be on yet another computer, such as an X Windows terminal or any other workstation running an X Windows server process. The display software is the server for purposes of analyzing



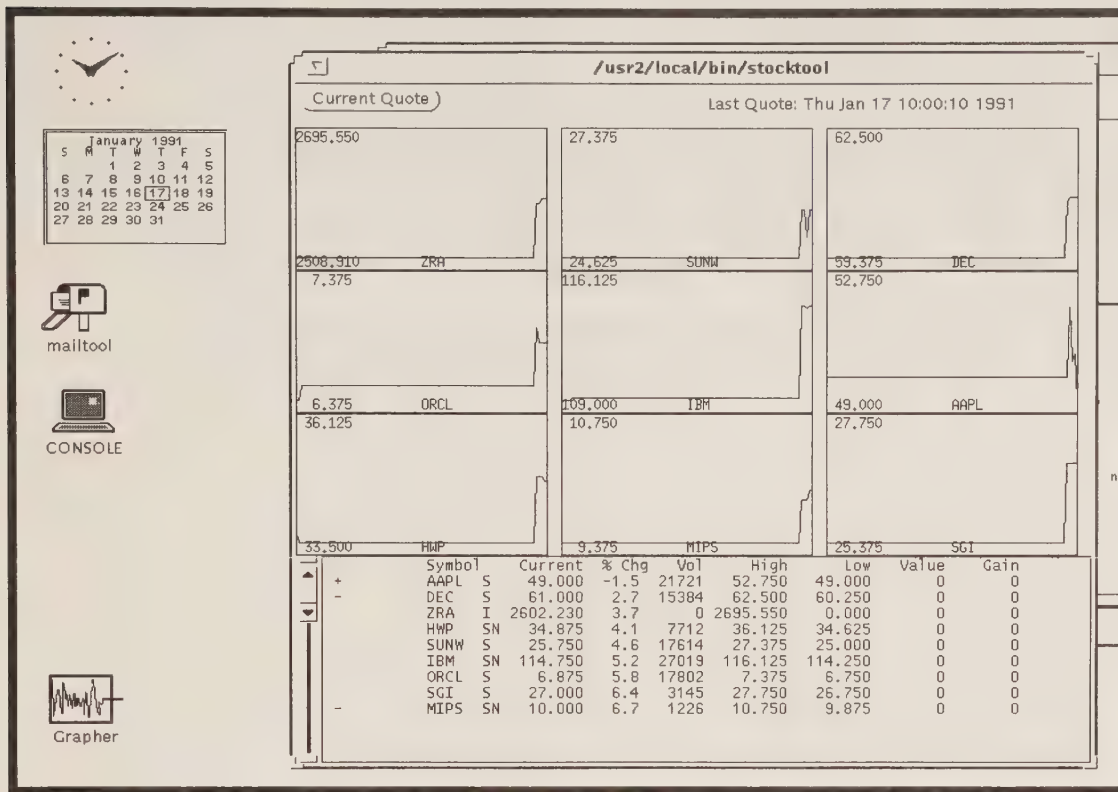
## 5-11 The Weather Tool

data, but becomes a client of the workstation's X Windows Server for purposes of writing the data on the screen.

The RPC protocol provides the underlying foundation that allows clients to submit remote procedure call requests across the network. That application program can then use other services such as the window system to display that information on the screen.

### *An RPC Example*

Figure 5-12 shows an example RPC application, this time a program written at Sun Microsystems. Sun, like many companies, likes to give its employees stock purchase options, a common way in large corporations to increase employee motivation. To allow the employee to concentrate on real work instead of tuning in to business reports on the radio, a clever programmer developed the RPC Stock Tool, shown in Figure 5-12. This tool uses the RPC



Courtesy of Sun Microsystems

5-12 The Sun Stock Tool

programs to obtain quotes on the performance of various stock prices. Notice in our example that the price of Sun stock is being compared to other computer companies.

The stock tool program uses RPC to get data but then places calls to the window system to display that data. It is not unusual at Sun for users to mount their window systems from one server and application programs from another.

Figure 5-12 also shows a few other RPC-based programs. Notice the calendar tool which allows users to coordinate their own calendars with those of others. The calendar also provides a reminder capability. Below the calendar is the mail tool. Notice that the little flag is up on the mailbox, indicating that new mail has arrived. Below the mail tool is a “grapher” icon, used by SunNet Manager to display network management information on the screen. SunNet Manager uses RPC to obtain that information from different managed nodes in the network.

### For Further Reading

John Corbin, *The Art of Distributed Applications—Programming Techniques for Remote Procedure Calls*. Springer Verlag (New York: 1991). The definitive look at the ONC RPC mechanism by one of Sun's senior engineers.

Sun Microsystems, *Network Programming Guide*, Part Number: 800-3850-10, Revision A of 27 March, 1990. Contains introductory material and protocol specifications to RPC, XDR specification and programming, and an RPC and rpcgen tutorial.

———, *RPC: Remote Procedure Call Protocol* (ver. 2). RFC 1057.

———, *XDR: External Data Representation Standard*. RFC 1014.



# Network File System



# Network File System

## NFS

When a file system is mounted on a local disk, the entire file system is mounted. Disks are made up of partitions and inside each partition is a file system. With NFS, the file system has been extended over the network. There is a difference, however: a portion of a remote file system can be mounted instead of the whole partition. Thus, the user can decide to take only the manual pages instead of the entire file system that would also include executables, user space, and other data.

NFS has three components on the server. The `exportfs` command is used to designate certain files as available to people on the network. The `export` file system command has several options so that systems can be made read-only, access limited to certain users, or even can allow remote root access.

The other two components are daemons on the server. The `rpc.mountd` daemon is specifically for handling the bootstrap problem: having a client notify the server it will be using files, which allows the server to authenticate the client. It also allows the server to return a file handle: a piece of data that the client will present on future requests so that the server knows which file is being referenced.

Why not have the file handle exchanged on each request? It is, but there needs to be a system-independent way of handling the initial reference. Inside the file handle the server hides information that it uses to locate the object. Presented with the file handle and a command (such as `read` objects in the directory), the file system returns a list of names. The lookup call is then used on the names to find their file handles. The mount protocol is a bootstrap for file handles: it allows different file systems with different naming syntaxes to all export file systems.

The third component is the NFS daemon itself. The NFS daemon handles incoming NFS calls and hands back answers. The daemon may handle many different users: remember that the protocol is stateless, meaning that

ro	Export as read-only (default is read-write).
rw=hostnames	Export as read-mostly: read-only to most hosts but read-write to the named hosts.
anon=uid	Requests from unknown users use the specified UID as the effective user ID. Root users (UID=0) are always considered "unknown" by the server unless they are included in the root option below. Default is -2 (nobody). Can disable anonymous access by setting to -1.
root=hostnames	Give root access only to the root users from the specified machines.
access=clients	Give mount access to each client listed (hostname or netgroup). Default value is to allow any machine to mount the directory.
secure	Require Secure RPC for accessing.

## 6-1 NFS Export Options

there is no information saved between requests and no connections. A typical server will have eight daemons running, meaning that up to eight simultaneous NFS operations can be processed.

### *Exporting File Systems*

The `exportfs` command is used to make a file system available to clients on the network. Exports can be performed manually by the superuser using the `exportfs` command:

```
/usr/etc/exportfs -a
```

This command takes all lines in the `/etc/exportfs` file and submits the information to the kernel. It is also possible to just export (or unexport) certain directories and to add any of the options on the command line that are not present in the `exportfs` file. On most servers, the command is executed once when the system boots.

A variety of options are available to limit (or enhance) how systems are exported (see Fig. 6-1). Read-only file access, for example, is often used on shared exported file systems like the manual pages that require no changes.

Other options such as `root` and `access` can be used when the information needs to be limited to certain groups of users. Root access is typically used to allow a diskless client root access to its root file system on a server. Particularly important in this regard is the anonymous export option, which sets what happens when an unknown user requests access.

Access control to exported file systems is implemented on a per client basis and export parameters allow a fine degree of control. Given a user



identification, the `exportfs` file will determine which file systems and options are allowed. Secure authentication of the user, an issue different from access control, is handled by the Secure NFS service.

### *Mounting File Systems*

Export is the way that a file system is available to the network. To actually mount a file system, the client needs to issue a mount command. There are three different ways that this mount command can be issued:

- As part of the workstation initialization from the file system table
- Interactively by a user with root privileges
- Using the Automounter

The Automounter is explained more extensively in Chapter 11. This service is a dynamic mounter and greatly simplifies the administration of NFS in large networks.

Normally, key file systems are mounted at boot time. A diskless workstation, for example, would need to mount operating system, root, and user file systems so that it could run Unix commands. Workstations would probably also want to mount file systems for the user's home directory, the manual pages, the common software systems that will be needed, and any other data that is used by the workstation.

Figure 6-2 shows information that can be included in the `/etc/fstab` file system table. Notice that a variety of different kinds of file systems may be mounted, and the list can be expanded. For NFS, there are a variety of options that control how the mount operation will proceed.

Many of the mount parameters are things that control operation of the local NFS client, such as how aggressively to do retries. This degree of control gives the workstation the capability to adjust parameters for better performance (or, in the wrong hands, to become a bad citizen of the network by being overly aggressive on retries).

The hard and soft mount options are quite important for clients. A soft mount means that after a server stops responding, the client will give up the connection and return an error to the application program. With a hard mount, the client just keeps on trying. Hard mounts are needed whenever a file system will have critical write operations, as in the case of the home directory.

A hard mount can cause problems when the target server is truly dead instead of just temporarily overloaded. If a hard mount is specified, it is often a good idea to also specify the interrupt (`intr`) option to allow the keyboard to stop an infinite repetition of retries.

Figure 6-3 shows an example of a small `/etc/fstab` file. Notice that the first two mounts are local disk drives. The last three mounts are each coming from a different server. The manual pages come from `server1`. This

File System Types	
4.2	4.2 bsd block-special device.
nfs	Exported NFS file system.
swap	Swap partition.
rfs	RFS file system.
tmp	File system in virtual memory.
hsfs	High Sierra Standard CD-ROM.
pcfs	DOS file system (diskette).
NFS Options	
bg fg	If first attempt fails, continue retrying in background (bg) or foreground (fg).
noquota	Prevent quota from checking to see if user over quota on this file system.
retry=n	Number of times to retry mount operation.
rsz=n	Read transfer buffer size.
wsz=n	Write transfer buffer size.
timeo=n	NFS timeout for retransmissions, to n tenths of second.
retrans=n	Maximum number of NFS retransmissions for each RPC request.
port=n	Server IP port number.
soft/hard	Return an error if server doesn't respond or continue the retry request until server responds.
intr	Allow keyboard interrupts on hard mounts.
secure	Use Secure RPC authentication.
agregmin=n	Hold cached attributes for n seconds after file modification. Minimum for regular files.
agregmax=n	Maximum value for regular files.
acdirmin=n	Minimum value for directories.
acdirmax=n	Maximum value for directories.
actimeo=n	Set minimum and maximum times for regular files and directories to n seconds.
noac	Suppress attribute caching.
DEFAULTS	fg, retry=10000, timeo=7, retrans=3, port=NFS_PORT, hard, agregmin=3, agregmax=60, acdirmin=30, acdirmax=60

## 6-2 Fstab NFS Options

```
# /etc/fstab
/dev/sd0a          /          4.2          rw           1           1
/dev/sd0g          /usr       4.2          rw           1           2
node1:/usr/man     /usr/man   nfs          bg,ro,soft,actimeo=3600 0           0
node2:/usr/news    /usr/news  nfs          bg,ro,hard,intr,secure  0           0
node3:/usr/local   /usr/local nfs          bg,ro,soft,actimeo=3600 0           0
```

### 6-3 Sample /etc/fstab File

mount is a background mount: if for some reason the mount doesn't immediately succeed, it will continue to retry but not prevent the user from proceeding with work.

The other two mounts are for news files from Usenet and for a local filesystem. Notice that the local filesystem is a read-only mount. The timeout for attributes, the frequency with which the client checks to see if a file has changed, is set to 1 hour since it is expected that files will not change very often.

The showmount command allows a user to see what systems are currently mounted on the client system. It is also possible to ask a server which systems are currently being exported. The answer can show all the hierarchies currently remotely mounted by clients (with the caveat that some of the clients may have crashed and the server may have slightly stale data).

Figure 6-4 shows the result of a showmount command. In the first command, the inquiry is to find out which file systems are being exported by the server named "commserv." For each of the exported file systems, there are only certain groups that are allowed to perform a mount operation.

The next showmount command uses the -a option, which shows all mounts currently active on a server. The listing in Figure 6-4 is only a small fraction of the mounts that were active on this host, a file server located in the engineering group at Sun.

Just as a server may export to a wide variety of clients, so may a client mount file systems from many different servers. The same mount command that is used to interactively mount a file system can be used to show the current status of a workstation (see Fig. 6-5). Notice that two different servers are providing files for the workstation hydramatic.

Just as a single server can handle many different workstations, so can the server mounts be for very large file systems. In Chapter 2, the network configuration for the Center for Numerical Aerodynamic Simulation (NAS), located at the NASA-Ames Research Center, was described. That network has two large Cray computer systems, connected to each other via an Ultra-net gigabit network. Each of these Cray computer systems runs Unix as well as the NFS protocols. The file systems on each system are exported to the other system.



hydramatic% showmount -e commserv

export list for commserv:

/usr	engineering,spd-topdomain
/export/home	engineering,spd-topdomain
/var/spool/mail	engineering,spd-topdomain
/export/exec/sun3.sunos.4.1BETA	engineering,spd-topdomain
/export/exec/kvm/sun3.sunos.4.1BETA	engineering,spd-topdomain
/export/root/majestic	majestic
/export/root/osento	osento
/export/swap/osento	osento

hydramatic% showmount -a commserv

alexander:/export/home  
alty:/export/home  
anacapa.Corp.Sun.COM:/export/home  
ark:/export/home  
atticus:/export/home  
baraka:/export/home  
bari:/export/home  
barry:/export/home  
batcomfs:/export/home/dcw  
bbeachco:/export/home  
bigboss.Corp.Sun.COM:/export/home  
billthecat:/export/home  
bisun:/export/home  
blizzard:/export/home  
blond:/export/home  
blueox:/home/commserv  
bunny2:/export/home  
bzone.Corp.Sun.COM:/export/home  
calsun:/usr/share/man  
centauri.Corp.Sun.COM:/export/home  
chance:/export/home  
chao:/export/home  
charm:/export/home  
cheers2:/export/home  
chest:/export/home  
chispa:/export/home  
clown:/export/home  
colorwriter:/home/commserv  
comedy:/export/home



```

hydramatic% mount -p
/dev/sd6a                /                4.2  rw  1 1
/dev/sd6g                /usr             4.2  rw  1 2
comm:/export/home        /usr/export/home/ouch  nfs  rw  0 0
comm:/export/snm         /ouch/snm        nfs  rw  0 0
swserver44:/export/local/sun3 /usr/local       nfs  ro  0 0
swserver44:/export/local/share /usr/local/share  nfs  ro  0 0

```

## 6-5 Locally Mounted Systems

Figure 6-6 shows the `df` and `mount` commands run on one of the systems, the CRAY YMP. Notice the large number of files that are being mounted from the host Navier, the CRAY 2 system. Running the same commands on the CRAY 2 shows a similar large number of mounts from the host Reynolds (the YMP). Notice for example the `/navier/src3` file system, which is made of 1.7 million 4-kbyte blocks, for a total mounted file system size of 6.8 Gbytes.

What is important at this point is that a client is able to store data on one or more servers throughout the network. To the local file system, the files appear to be locally mounted. If the two systems are both based on Unix, the mapping is fairly straightforward.

This service works for other operating systems. For a PC, for example, a file system can be mounted using PC-NFS and appear as additional disk drives, the appropriate semantic for the PC. For a VMS system, NFS mounts can also be performed, again appearing as a new device.

## The NFS Protocol

Once the mount service has performed its task of providing a file handle to the client, the NFS service takes over. The NFS protocol is stateless: all requests are independent from any other request.

This means, among other things, that no session is maintained between the client and the server. Usually, the UDP protocols are used so there is not even a transport level connection. It is possible to run NFS over TCP protocols, but the basic LAN implementation is connectionless.

The purpose of a stateless protocol is to simplify crash recovery. If every request is independent from any other, it is possible to lose a client or server without going through a crash-recovery procedure.

Lack of state just means that if a recovery is needed, it will be up to the client to perform that operation. If a server crashes and then recovers, NFS requires no recovery procedure to start backup.

The basic operations that the NFS protocol supports are simply an outgrowth of the services one would expect on a local file system. Figures 6-7

Script started on Wed Nov 28 13:52:41 1990 (Cray YMP)

(/dev/tty105)

reynolds.lkoke 31 df

/navier/usr	(navier-ulp:/usr):	21772 4K blocks ( 7.3%)	
/navier/scr4	(navier-ulp:/scr4):	797848 4K blocks ( 30.9%)	
/navier/scr3	(navier-ulp:/scr3):	1706166 4K blocks ( 66.1%)	
/navier/scr2	(navier-ulp:/scr2):	1051558 4K blocks ( 40.7%)	
/navier/scr1	(navier-ulp:/scr1):	1518537 4K blocks ( 58.8%)	
/navier/u/ng	(navier-ulp:/u/ng):	7802 4K blocks ( 2.6%)	
/navier/u/ne	(navier-ulp:/u/ne):	222234 4K blocks ( 24.9%)	
/navier/u/nd	(navier-ulp:/u/nd):	301989 4K blocks ( 50.8%)	
/navier/u/nc	(navier-ulp:/u/nc):	268600 4K blocks ( 30.1%)	
/navier/u/nb	(navier-ulp:/u/nb):	114466 4K blocks ( 13.1%)	
/navier/u/na	(navier-ulp:/u/na):	482460 4K blocks ( 54.1%)	
/proc	(/proc):	108297 4K blocks ( 42.7%)	566 procs
/51root/usr/src	(/dev/dsk/src):	130113 4K blocks ( 29.2%)	
/51root/usr	(/dev/dsk/usr):	51508 4K blocks ( 17.3%)	
/51root	(/dev/dsk/root):	108252 4K blocks ( 72.6%)	
/usr/src	(/dev/dsk/60src):	156279 4K blocks ( 38.0%)	18537 I-nodes
/scr8	(/dev/dsk/scr8):	644989 4K blocks ( 25.1%)	
/scr7	(/dev/dsk/scr7):	1169864 4K blocks ( 45.5%)	
/scr6	(/dev/dsk/scr6):	567802 4K blocks ( 22.1%)	
/scr5	(/dev/dsk/scr5):	1822596 4K blocks ( 70.9%)	
/u/rf	(/dev/dsk/rf):	161337 4K blocks ( 27.1%)	
/u/re	(/dev/dsk/re):	627650 4K blocks ( 52.8%)	92220 I-nodes
/u/rd	(/dev/dsk/rd):	834365 4K blocks ( 70.2%)	110770 I-nodes
/u/rc	(/dev/dsk/rc):	567311 4K blocks ( 47.7%)	91135 I-nodes
/u/rb	(/dev/dsk/rb):	86984 4K blocks ( 9.7%)	82841 I-nodes
/u/ra	(/dev/dsk/ra):	820870 4K blocks ( 69.1%)	105061 I-nodes
/core	(/dev/dsk/core):	351213 4K blocks ( 78.7%)	65525 I-nodes
/usr/spool	(/dev/dsk/spool):	42737 4K blocks ( 37.3%)	27943 I-nodes
/wrk	(/dev/dsk/wrk):	708035 4K blocks ( 59.6%)	166903 I-nodes
/usr	(/dev/dsk/60usr):	59623 4K blocks ( 20.1%)	18223 I-nodes
/tmp	(/dev/dsk/tmp):	586476 4K blocks ( 98.7%)	65046 I-nodes
/	(/dev/dsk/60root):	100229 4K blocks ( 67.2%)	

reynolds.lkoke 33 /etc/mount

```

/ on      /dev/dsk/60root read/write on Mon Nov 26 06:24:52 1990
/tmp on   /dev/dsk/tmp   read/write,rw on Mon Nov 26 06:25:03 1990
/usr on   /dev/dsk/60usr read/write,rw on Mon Nov 26 06:25:04 1990
/wrk on   /dev/dsk/wrk  read/write,rw,quota=/etc/quota60/wrk
                        on Mon Nov 26 06:25:05 1990
/usr/spool on /dev/dsk/spool read/write,rw,quota=/etc/quota60/spool
                        on Mon Nov 26 06:25:06 1990
/core on   /dev/dsk/core read/write,rw on Mon Nov 26 06:25:06 1990
/u/ra on   /dev/dsk/ra   read/write,rw,quota=/etc/quota60/home

```

		on Mon Nov 26 06:25:07 1990
/u/rb on	/dev/dsk/rb	read/write,rw on Mon Nov 26 06:25:08 1990
/u/rc on	/dev/dsk/rc	read/write,rw,quota=/dev/dsk/ra
		on Mon Nov 26 06:25:09 1990
/u/rd on	/dev/dsk/rd	read/write,rw,quota=/dev/dsk/ra
		on Mon Nov 26 06:25:10 1990
/u/re on	/dev/dsk/re	read/write,rw,quota=/dev/dsk/ra
		on Mon Nov 26 06:25:10 1990
/u/rf on	/dev/dsk/rf	read/write,rw on Mon Nov 26 06:25:11 1990
/scr5 on	/dev/dsk/scr5	read/write,rw,quota=/etc/quotab0/scratch
		on Mon Nov 26 06:25:12 1990
/scr6 on	/dev/dsk/scr6	read/write,rw,quota=/dev/dsk/scr5
		on Mon Nov 26 06:25:12 1990
/scr7 on	/dev/dsk/scr7	read/write,rw,quota=/dev/dsk/scr5
		on Mon Nov 26 06:25:13 1990
/scr8 on	/dev/dsk/scr8	read/write,rw,quota=/dev/dsk/scr5
		on Mon Nov 26 06:25:14 1990
/usr/src on	/dev/dsk/60src	read/write,rw on Mon Nov 26 06:25:14 1990
/51root on	/dev/dsk/root	read/write,rw on Mon Nov 26 06:25:15 1990
/51root/usr on	/dev/dsk/usr	read/write,rw on Mon Nov 26 06:25:16 1990
/51root/usr/src on	/dev/dsk/src	read/write,rw on Mon Nov 26 06:25:16 1990
/proc on	/proc	read/write on Mon Nov 26 06:25:17 1990
/navier/u/na on	navier-uid:/u/na	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:18 1990
/navier/u/nb on	navier-uid:/u/nb	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:19 1990
/navier/u/nc on	navier-uid:/u/nc	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:19 1990
/navier/u/nd on	navier-uid:/u/nd	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:20 1990
/navier/u/ne on	navier-uid:/u/ne	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:21 1990
/navier/u/ng on	navier-uid:/u/ng	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:21 1990
/navier/scr1 on	navier-uid:/scr1	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:21 1990
/navier/scr2 on	navier-uid:/scr2	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:22 1990
/navier/scr3 on	navier-uid:/scr3	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:22 1990
/navier/scr4 on	navier-uid:/scr4	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:23 1990
/navier/usr on	navier-uid:/usr	read only,nosuid,bg,soft
		on Mon Nov 26 06:43:23 1990
reynolds.lkoke 34 exit		
reynolds.lkoke 35		



SUMMARY	Delta T	DST	SRC
M 1		Intrln0027C0+DEC	029487 NFS C Lookup hostinfod.o in F=05
2	0.0390	DEC 029487+Intrln0027C0	NFS R No such file
3	0.0089	Intrln0027C0+DEC 029487	NFS C Create hostinfod.o in F=05
4	0.0377	3Com 063841+3Com 115176	Telnet C PORT=4704 U
5	0.0419	3Com 115176+3Com 063841	Telnet R PORT=4704 U
6	0.0048	3Com 063841+3Com 115176	TCP D=23 S=4704 ACK=73829906
7	0.1037	DEC 029487+Intrln0027C0	NFS R OK F=5D19
8	0.2407	Intrln0027C0+3Com 063841	DNS C ID=3 OP=QUERY NAME=
9	0.0506	3Com 063841+Intrln0027C0	DNS C ID=33629 OP=QUERY NAME=

DETAIL

NFS: ----- SUN NFS -----

NFS:

NFS: Proc = 4 (Look up file name)

NFS: File handle = 04090000091000000800000000000000

NFS: 00000000000000000000000000000000

NFS: File name = hostinfod.o

NFS:

NFS: [Normal end of "SUN NFS".]

NFS:

Frame 1 of 300

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 6-7 NFS Lookup Operation

SUMMARY	Delta T	DST	SRC
53	0.0657	DEC 029487+Intrln0027C0	NFS R OK 42 entries (No more)
54	0.0236	Intrln0027C0+3Com 063841	DNS C ID=0 OP=QUERY NAME=LOTS.A.S
55	0.0170	Intrln0027C0+DEC 029487	NFS C Lookup getkey.c in F=0519
56	0.0301	DEC 029487+Intrln0027C0	NFS R OK F=591B
57	0.0347	3Com 063841+Intrln0027C0	DNS R ID=0 STAT=OK NAME=LOTS.A.ST
58	0.2200	Intrln0027C0+DEC 029487	NFS C Lookup cc in F=0519
59	0.0397	DEC 029487+Intrln0027C0	NFS R No such file
60	0.0152	3Com 063841+3Com 115176	Telnet C PORT=4704 <0D><0A>
61	0.0232	Intrln0027C0+DEC 029487	NFS C Lookup bin in F=FA79

DETAIL

NFS: ----- SUN NFS -----

NFS:

NFS: Proc = 4 (Look up file name)

NFS: Status = 0 (OK)

NFS: File handle = 040900005C1200000100000000000000

NFS: 00000000000000000000000000000000

NFS: File type = 1 (Regular file)

NFS: Mode = 0100644

NFS: Type = Regular file

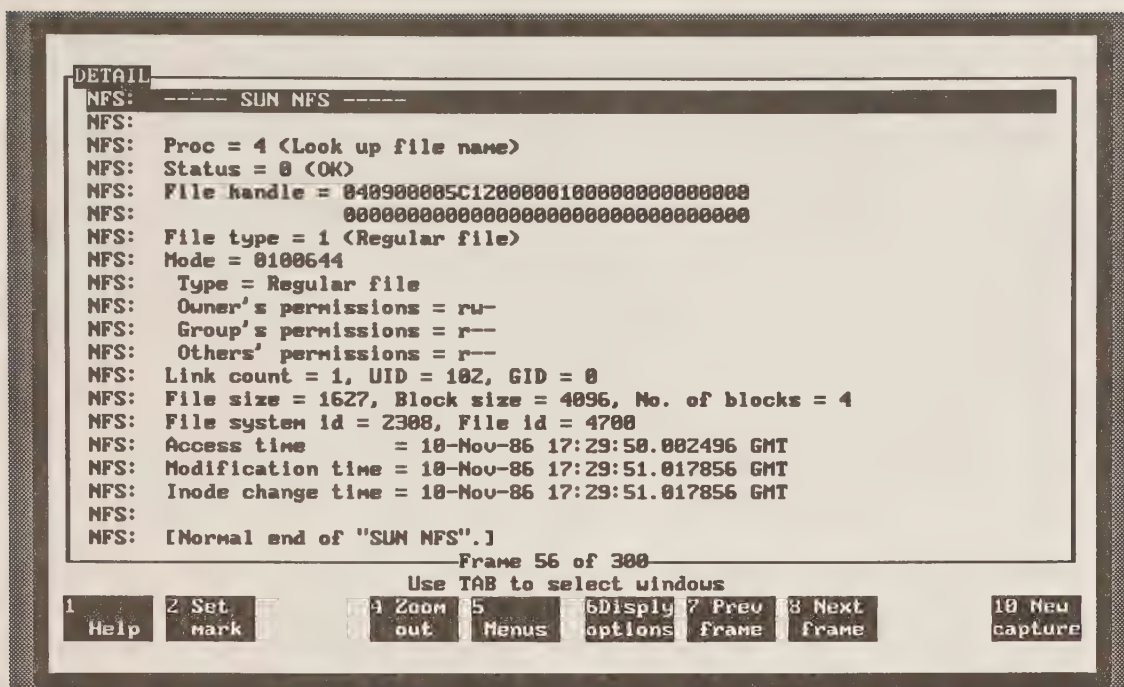
Frame 56 of 300

Use TAB to select windows

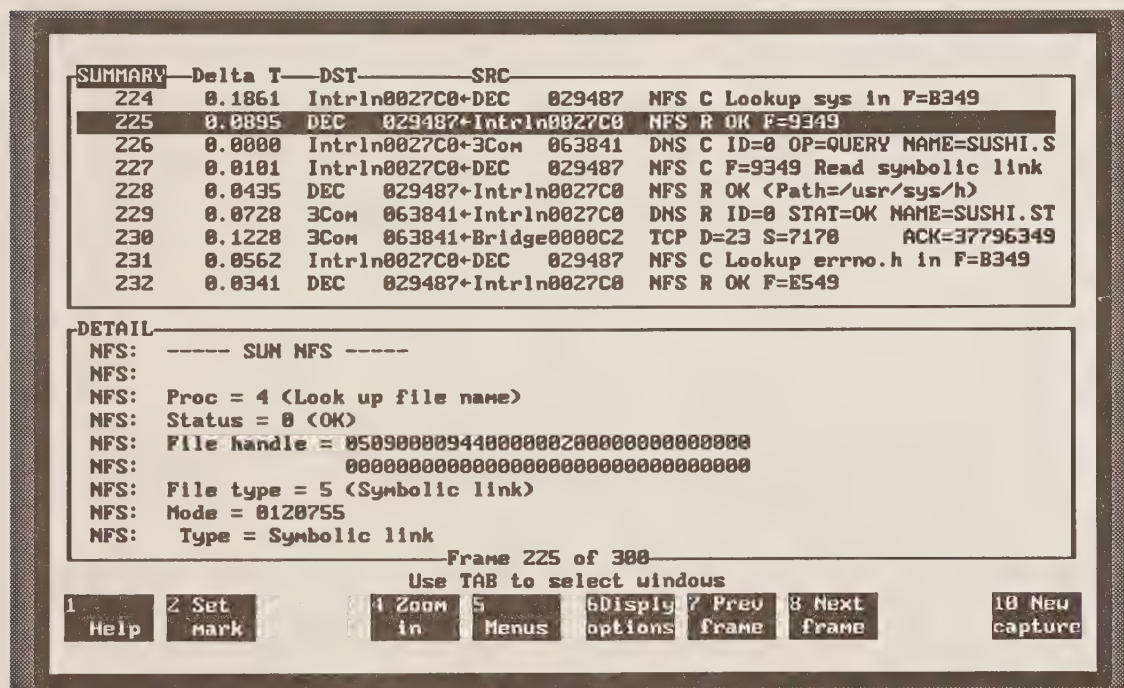
1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 6-8 Lookup Traffic





## 6-9 Lookup Reply



## 6-10 Following a Symbolic Link

and 6-8 show the basic lookup operation. A client submits a file name and asks for a lookup on that file name. The lookup returns a file handle, the basic currency used to identify objects for future read and reply operations.

The reply to any lookup command is a lookup reply. This reply gives the user back various attributes that are kept for a file. Figure 6-9 shows more detail on the lookup request for the file `getkey.c`. Notice that the file attributes are very Unix-like, including the permission mode bits we expect to see with Unix files.

An important attribute returned for a file is the modification time. This data is used to determine if a cached version of a file is the same as the file on the server. When the file system gets a file, it often caches the file for the user, along with the last modification time. If a user wishes to change the data, the workstation can compare the last modification time in cache with that on the server to see if anybody else has made changes in the meantime.

A lookup on a file may return information other than a regular file. For example, in Figure 6-10 a lookup command has returned an attribute indicating that this file is really a symbolic link to another file. The symbolic link is returned to the client kernel, which will then typically issue a read-link call to trace the link to another file (which may also be a link). In this way, symbolic links are traced until the real file is reached.

Figures 6-11 and 6-12 show another type of lookup operation. In Figure 6-11, the workstation is issuing a get attributes call to the NFS server. We can see the response in the next packet was a successful return. The client then followed this up with another lookup for a particular file located inside that directory.

The lookup command generates a file handle for a file, which can then be used by the client system to issue a read request. Figure 6-13 shows a read request. Notice that this packet is one NFS call with some 3000 bytes of data. At the bottom of the pack there is an indication that the packet has 1360 bytes of data but there are 1712 bytes missing (i.e., still to come in the next packet).

NFS uses RPC calls to get data over the network. RPC, in turn, uses the underlying UDP or TCP transport services. Sun's UDP implementation allows message sizes of eight kbytes long, whereas for TCP the stream-like interface allows an infinite "message" size. (TCP packets have a limitation of 64 kbytes but the interface to TCP is simply a stream of data.)

Figure 6-14 shows what has happened to the "missing" data. Notice that the first NFS packet is sent, followed by two UDP continuations. These UDP continuations have the rest of the data that RPC uses to piece a reply together before presenting the complete reply to the workstation or server. The UDP continuation is totally transparent to upper level users—UDP will

reassemble incoming IP fragments into a complete packet (though it will only do so on a best-effort basis with no delivery guarantees).

While UDP gives the appearance of an 8-kbyte datagram service, the underlying network maximum transfer unit (MTU) is important. When a UDP packet is fragmented, and one of the fragments is lost, then the whole UDP packet is lost. This means that the upper-level user, RPC in this case, would retransmit 8 kbytes of data, even though a 576-byte IP packet was lost. When NFS moves into a wide-area environment, the service provided by the transport layer and the MTU of the underlying network become increasingly important since the links are more complicated and less reliable.

One advantage of using a TCP-based transport service for NFS is that TCP provides the delivery guarantees. Retransmission of data is not at the RPC level, but at the TCP fragment level. Since TCP is adaptive to performance on the underlying network, it makes better use of congested facilities than UDP does.

Note that NFS is not totally ignorant of the underlying network. Many client implementations use an adaptive retransmission method so that retries are not blindly sent down every time a request times out. The amount of time grows between each retry, reducing the network load from retransmissions in times of network congestion or failure. This adaptive retransmission also adjusts the packet sizes so smaller packets are sent in times of scarce network resources.

Some clients do not implement adaptive retransmission. Here, the client simply waits for the retry interval, which is a default of 0.7 seconds, and retries. The client then takes the current interval and multiplies it by 4 and retries again, until the ceiling value of 60 seconds is reached. The number of retries within a given timeout cycle is configurable for each mount point. In the NFS reference port, the default number of retries in a given timeout cycle is three.

## The NFS Implementation

The provider of the NFS service are the NFS Server Daemons (NFSD). A Unix system can have several of these. If a system has eight daemons, eight requests can be processed concurrently. On Unix clients, there is the optional (though universally implemented) Block I/O Daemon (BIOD). These daemons read data before a program requests it (asynchronous read ahead) and also perform write behind.

The write behind on the BIOD means that when a program writes to a file, the client program's write command responds as soon as the data gets into the cache. The data is still not in the server yet. However, when the program does a close on a file, that call will not return until all data has been flushed through to stable storage.



SUMMARY	Delta T	DST	SRC	
129	0.0391	3Com	115176+3Com	063841 Telnet R PORT=4704
130	0.0046	3Com	063841+3Com	115176 TCP D=23 S=4704 ACK=73829909
131	0.1153	3Com	063841+3Com	115176 Telnet C PORT=4704
132	0.0392	3Com	115176+3Com	063841 Telnet R PORT=4704
133	0.0048	3Com	063841+3Com	115176 TCP D=23 S=4704 ACK=73829909
134	1.1557	Intrln0027C0+3Com	063841	DNS C ID=3 OP=QUERY NAME=
135	0.5587	Intrln0027C0+DEC	029487	NFS C F=420E Get attributes
136	0.0296	DEC	029487+Intrln0027C0	NFS R OK
137	0.0091	Intrln0027C0+DEC	029487	NFS C Lookup c2 in F=420E

DETAIL

NFS: ----- SUN NFS -----

NFS:

NFS: Proc = 1 (Get file attributes)

NFS: File handle = 00090000420700000000000000000000

NFS: 00000000000000000000000000000000

NFS:

NFS: [Normal end of "SUN NFS".]

NFS:

----- Frame 135 of 300 -----

Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

6-11 Get Attributes Request

DETAIL

NFS: ----- SUN NFS -----

NFS:

NFS: Proc = 1 (Get file attributes)

NFS: Status = 0 (OK)

NFS: File type = 2 (Directory)

NFS: Mode = 040755

NFS: Type = Directory

NFS: Owner's permissions = rwx

NFS: Group's permissions = r-x

NFS: Others' permissions = r-x

NFS: Link count = 2, UID = 0, GID = 10

NFS: File size = 512, Block size = 8192, No. of blocks = 2

NFS: File system id = 2304, File id = 1858

NFS: Access time = 10-Dec-86 12:45:52.062496 GMT

NFS: Modification time = 20-Oct-86 22:45:50.039572 GMT

NFS: Inode change time = 24-Oct-86 22:40:52.001248 GMT

NFS:

NFS: [Normal end of "SUN NFS".]

NFS:

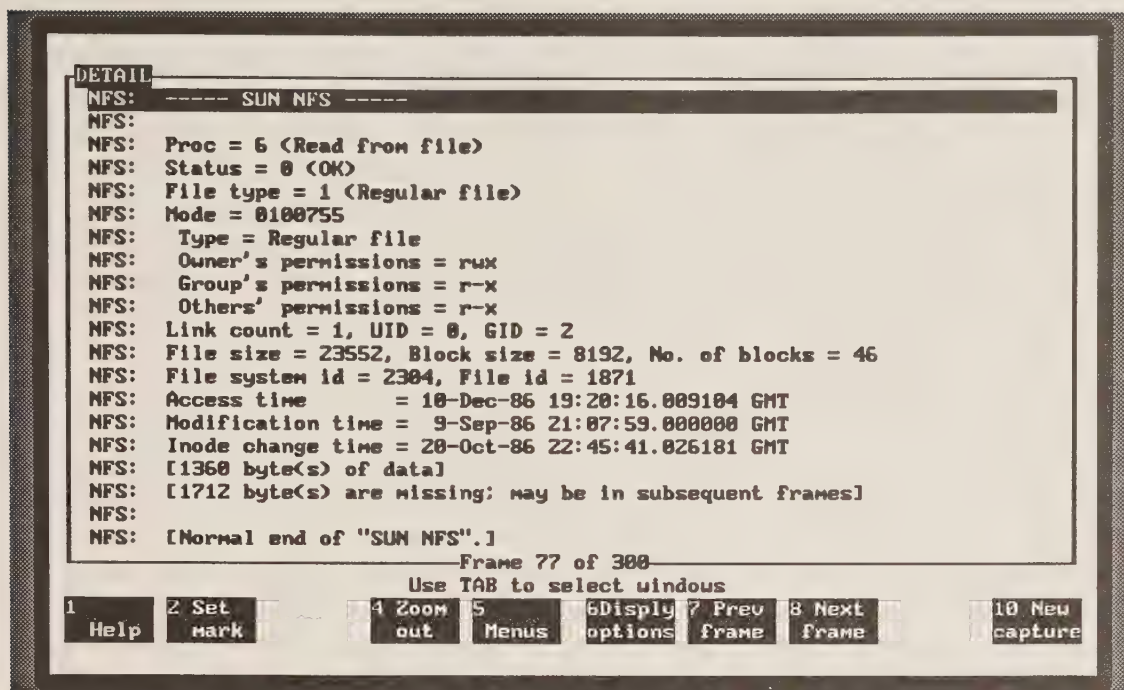
----- Frame 136 of 300 -----

Use TAB to select windows

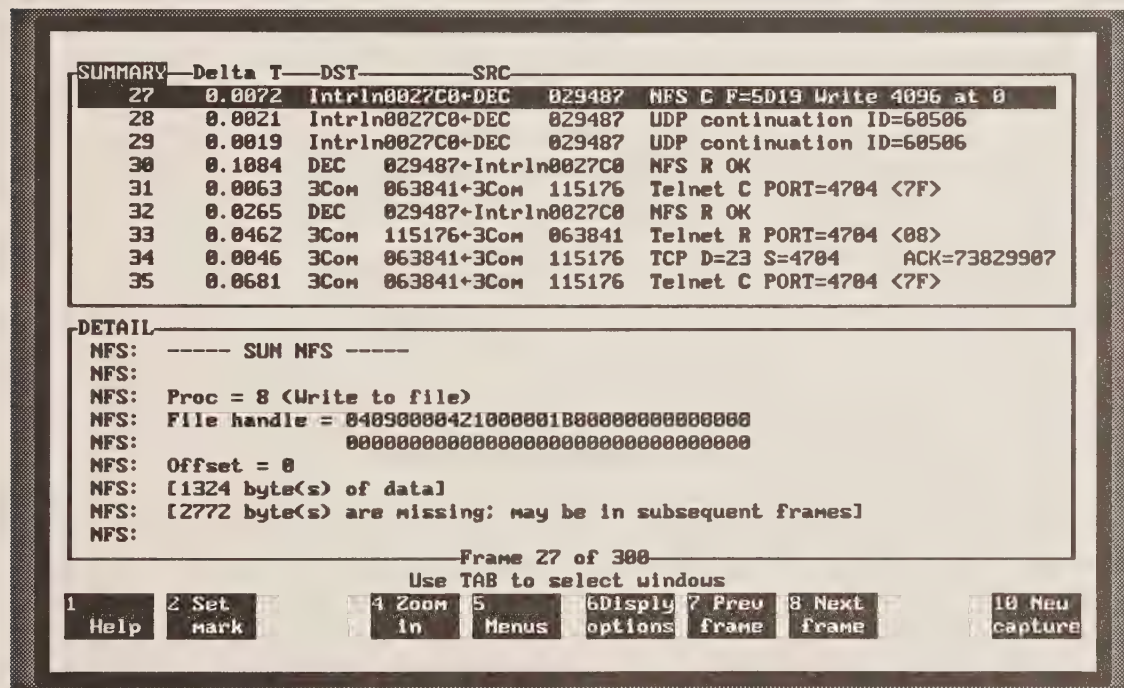
1 Help	2 Set mark	4 Zoom out	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	------------	---------	-------------------	--------------	--------------	----------------

6-12 NFS Get Attributes Response





## 6-13 NFS Read Operation



## 6-14 NFS Write with UDP Continuations

The number of BIODs certainly affects the performance of the client and the server. More BIODs, up to a point, increases performance on the client, particularly when many write operations are being performed.

### *Adding State*

The statelessness of the NFS protocol is nice for crash recovery. But, there are times when some state can help increase performance. In some work performed at Digital and Sun, a bit of state was added to the server to dramatically increase performance.

The basic idea was to find duplicate NFS requests which cause extra work on the server. Duplicate requests can arise from delays caused by slow servers or congested networks (which result in lost replies). Idempotent operations can be reapplied in a stateless environment. Nonidempotent requests cannot, in general, be reapplied non-destructively. A read is an example of an idempotent request. A file block can be read repeatedly with no ill effect. A create call is deemed non-idempotent because re-execution results in an error. Avoiding those duplicate requests also has the nice side effect of speeding up performance.

If a server is slow, a client may send multiple requests because of retransmissions. At best, this uses server resources. At the worst, it produces incorrect results. NFS writes are the most costly NFS server operation. If a server is under a heavy write load, it often has the double whammy of processing duplicate write requests.

A single process may have several outstanding read and write requests, especially if the client is running BIODs. These processes perform client read-ahead and write-behind functions asynchronously on behalf of other processes on the client. Each individual request may timeout individually and be retransmitted.

There is not much the system administrator can do to handle retransmission problems from the server. We depend on clients to control their request loads as the server becomes loaded. There is no requirement, however, that a given client be kind. Typical client implementations let a user run as many BIODs as he or she wants and to mount with arbitrarily small retransmission timeout values.

Of the sixteen operations in version 2 of NFS, nine are nonidempotent: create, remove, link, symlink, mkdir, rmdir, rename, setattr, and write. The first seven are pretty obvious. If a file is created the first time, the second application of that create operation will fail (with a "file already exists" message). The last two, set attributes and write, are a little less obvious. If they are applied immediately, they are idempotent: they yield the same results.

What happens, however, if they are reapplied after an intervening operation? In this case, the results can be a bit peculiar. For example, the set attribute command can be used to truncate a file. We truncate the file

once, somebody else adds some data, and then it is truncated a second time. Whoops.

Avoiding repeating requests is a part of the original kernel RPC reference implementation, which keeps a cache of recent transaction IDs (xids). Transaction IDs for nonidempotent operations are kept in this cache. When an operation succeeds, there is no need to consult the cache. If an operation fails, however, the cache is checked to see if the failure was caused by a duplicate transaction. This standard behavior could be found in all NFS implementations: failure of create, remove, link, mkdir, and rmdir resulted in a check on the cache.

The operation of this duplicate ID check was changed in two ways. First, it was expanded to include all operations. Second, a check is made before doing the operation instead of afterwards.

Chet Juszczak of Digital Equipment worked on this change and documented an implementation of these changes on some experiments with two different VAX servers. It was found that smaller servers tended to get more duplicate writes, since they were more likely to run out of processing power and to thus not respond quickly enough.

With an aggressive client (a MicroVAX II) and a small server (also a MicroVAX II), 1 Mbyte of data was written to a file, using 128 requests of 8 kbytes each. This file was always empty when the write started. In the case of the small server, the client ended up generating 190 write requests, of which 62 were duplicates. The server processed 61 of those extra 62 write requests, giving it an effective write speed of 35 kbytes per second.

On larger processors, fewer write requests are duplicated because the server is more likely to have processing power available. On a VAX 8550, for example, only 141 write requests were generated, of which 13 were duplicates. The effective write speed went up to 70 kbytes per second with 20 iterations.

Next, the scope of the RPC transaction cache was increased to add all request types. An "in progress" flag was added to each cache entry, as well as a transaction completion code, and a timestamp. The cache was also used differently. The new processing made a cache check one of the very first actions in servicing an NFS request. If the request is in cache and marked in progress, the request is just discarded with no response. If a request is not in cache, it is added and marked as being in progress.

If a transaction is in the cache, but not in progress, the mechanism is slightly different. If the duplicate is within a throwaway window (typically 3 to 6 seconds), the packet is dropped without a response. After the throwaway window, the server performs the request. This is to solve a common problem of an initial reply passing a retransmitted request in the network.

Finally, special treatment was added for the nonidempotent calls `setattr` and `write`. For these operations, completion status and a timestamp are



kept when the operation is completed. If the operation completed successfully, the cache entry was updated (including the timestamp for when the file was last changed). If a request is received that succeeded earlier, and the timestamp indicates that the file has not changed since the timestamp, a success response is sent to the client but the operation is not performed. However, if the cache entry has been flushed by the time the duplicate request is received, the operation is performed again.

The results of these modifications? All duplicate writes were eliminated from being processed. On the small server the write speed went from 35 to 48 kbytes/second. On the large server the speed went from 75 to 83 kbytes/second.

What is important here is not necessarily the clever use of the cache, but the fact that state was added to the server while still preserving the stateless nature of the protocol. The stateless protocol means very easy crash recovery. Duplicate request checks mean that not only is performance increased under load, but the integrity of the data is better preserved.

### *State*

There are a variety of other kinds of stateful information that can be kept by NFS implementations while still preserving the stateless protocol. Obviously, data in files on the server represent persistent state. This state is not bad, it makes NFS useful. When people term NFS stateless, they refer to the lack of an additional control state in the protocol and server. On a client, for example, server file handles, necessary for obtaining objects, are cached. Caching the file handle means the client is not forced to re-lookup the objects contained in the parent directory.

Clients can also maintain data and attribute caches. The BIOD on the client performs an asynchronous read ahead, fetching data before the client asks for it. This data grows stale in the cache, so programs always check to see that the data is the same before writing it again.

Servers also maintain state, for good performance. A typical example of stateful operation is a read-ahead cache. Adding state to a server is system dependent and not specified in the NFS protocol.

The server has flexibility on how to export resources. The administrator can assign rights on lists of machines and netgroups (a group of machines). The administrator can also require DES authentication by clients (known as Secure NFS).

Note that file access is inherently stateful. The way that a server stays somewhat stateless is by passing any state needed to perform an operation back to the client. A typical example is passing the file handle back to the client. The client treats the handle as an opaque data structure. This means that the server can hide whatever state information it wants inside the handle.



An example of the file handle is the one returned by a SunOS-based server. In this case, there are three pieces of information: the file system ID, the inode number, and the generation number.

## NFS on the MVS Operating System

To show that the NFS protocols apply to more than just a Unix environment, it is useful to look at the implementation of NFS on IBM's MVS operating system. Running many user applications on IBM's MVS operating system is kind of like using a moving van to deliver pizzas. However, there are instances where the power of the IBM mainframes can work quite well with a large corporate network. The IBM mainframe makes an excellent file server and certainly handles high-volume transactions processing well.

While the mainframe may be a perfect data vault, the workstation is certainly a better terminal than the IBM 3270. This is in fact exactly how Sun handles their most sensitive data. Purchasing and MRP systems for manufacturing are kept on IBM-compatible mainframes. Using SNA gateway products (see Chapter 16), the workstations are able to mount MVS file systems and access the corporate data.

The MVS/NFS implementation was jointly developed by Sun and Electronic Data Systems (EDS). The implementation is for a server MVS only: having a mainframe mount a PC's disk drive is not a service provided here.

The basic goal of the MVS/NFS implementation is to make the MVS data available to the Sun or other NFS client. The client might be a PC, a VAX, a Sun, or a Macintosh. Bridging the two environments is done via a gateway. The edge of the Sun network is a gateway; the edge of the mainframe is a channel-attached Front End Processor (FEP).

Access to the mainframe is accomplished using a Sun-based FEP. This software includes the TCP/CTCA translation and a program to forward RPC calls to the mainframes. The FEP is simply a Sun workstation, a SunLink Channel Adapter, and software. The FEP either shares an Ethernet with workstations or uses internetwork routing to provide services over WAN links. The RPC forwarder does socket creation and service registration on behalf of the servers on the mainframe and does RPC packet routing from the network to the registered service on the mainframe.

Clients accessing an NFS service use a different mount protocol called `mvsmount`. The normal mount protocol could not be used because of the introduction of the FEP into the chain and the need for login and password transmission to the MVS server.

The MVS RPC library is a standard implementation of RPC: any applications can be written (or ported) to use the RPC interface. One of the pre-written applications is NFS. For performance and security reasons, NFS is isolated from other applications as a separate MVS task.

On the gateway, the RPC forwarder performs the bulk of the work. On the mainframe, the counterpart is the channel task. This task communicates with the RPC forwarder across the channel hardware adapter. A special protocol used by the two allows service registration and deregistration, a channel reset capability, and blocking together of multiple RPC packets and control information in order to reduce the overhead of interrupts and channel set-up.

### *NFS Server Task*

The NFS server task is multitasking and multithreaded. Multiple copies of the NFS processing task, consisting of shared code but different data and stack regions, can execute concurrently within a single address space.

The basic problem the user and implementor find here is that NFS and MVS have very different file systems (see Fig. 6-15). MVS is a set of record-oriented data sets using EBCDIC encoding. NFS uses a byte stream with no records, and the character set is ASCII by default.

The differences don't stop there. MVS has attributes associated with data sets. Many of these attributes don't map into NFS attributes, and different types of data sets may have their own types of attributes. To make it even more interesting, a data set may have different attributes depending on the type of software that is installed. The Access Control Facility 2 (ACF2), for example, adds extra attributes.

Other major differences exist. NFS has a very strong notion of a hierarchical file system. MVS has a very weak notion of hierarchy: data sets are stuck in a flat filesystem.

Because MVS is a record-oriented file system, it delimits text lines as records. In the NFS world, there is no difference between text and binary as far as the server is concerned. Interpretation of the contents of a file is up to clients. Also, MVS has keyed access, which is missing in NFS.

The differences between records and files isn't limited to just access: the size of the files may actually be different. In MVS data sets, the size of the file is not immediately available. On some file types, all that is returned is the number of records (or even disk tracks). Even then, there is the problem of record packing within blocks and partial blocks—even if a size is returned, it may not be right.

This is a problem because NFS refers to file contents by the byte offset and length of the block required. If the size of the file or the location of all records is not known, it is hard to satisfy the NFS requests. The worst case is that the NFS server must read through a whole file, calculating offsets, just to be able to satisfy a read attributes request.

Because of the record structure, the issue arises of how to read a data set: binary or text. If binary, MVS/NFS does no translation of data stream con-

MVS	NFS
Record oriented	Byte stream
Complex rule-based security	User/group/world permissions
EBCDIC	ASCII (sort of)
Weak hierarchy for files	Implied hierarchy on file system
Very weak mapping of operations to a file hierarchy	Clearly defined hierarchical operations
Text lines delimited as records	Client implies text or binary
Defines structured files	No support for keyed access files

### 6-15 MVS and NFS File Systems

tents. In text mode, the data in the data set is assumed to be EBCDIC and is translated to ASCII when read from the network.

Since text and binary are different, when a user requests a size of a file from the server, the size will differ depending on the mode requested. A “text” file is one in which carriage returns have been added at the end of each record, meaning that the file will be longer than if read in simple binary mode.

To handle all these issues, there needs to be a way to tell the mainframe what we’re interested in seeing for data set types and for processing of attributes. This can be done on three levels:

- With a site-specific parameter block
- At mount point time
- At file access creation time

The site default is done via a text file that is read at server initialization. This defines a base set of defaults. Typical lines in the file might be:

```

mintimeout (1)
maxtimeout (1800)
timeout (90)
logout (1800)
# define primary and secondary allocations of space
# then define the number of blks (could do trks or cyls)
space (100,200), blks, blksize (6160), recfm (vb)
nfstasks(8)
```

At mount time, the attributes are specified in the mount command:

```
user% mount 'savant:sun.beeey,text,crlf,space(10,20)' /mnt
```



Note that the file specification includes attributes. The attributes are all separated by commas. The same overrides can be performed at file access time:

```
user% vi '/mnt/file.txt,space(256,256),recfm(vb),lf,dsorg(ps)'
```

This method exploits the fact that Unix doesn't care what is in a file name component and passes it on. PC clients, however, cannot do this since DOS tries to parse file names.

In all these commands, the user is indicating what type of file is desired and how to organize the data. In the file access example, the user is using the SunOS vi editor on an MVS file, which specifies how much space to allocate to the file, to use line feeds at the end of lines, and various other formats.

The original MVS/NFS server can handle four types of files on the mainframe:

- Sequential
- Direct access
- Partitioned
- VSAM

Within these files, the server can handle three record formats: fixed-length records, variable-length records, and undefined (binary). The server can also handle blocked and spanned records. A separate RPC service handles keyed access data sets (but not via NFS functionality).

A subsequent release of MVS/NFS included support for a true hierarchical file system on MVS with fully support for Unix names and attributes. This enables MVS to act as a full file server in a Unix NFS environment. Adding a full hierarchical file system to the pre-Columbian (i.e., flat) view defined by MVS is the subject of a forthcoming Sun Technical Report.

### *Concurrent Access*

Concurrent access to a mainframe data set from both the network and the mainframe is a problem in MVS. Typically, a data set has exclusive access until closed (at least if exclusive access is requested). The problem is this: if somebody mounts a read/write file system, the MVS server is going to take an exclusive open on the file.

But, NFS is stateless. How do we know when the user is done? There is no close in NFS. While the lack of a close operation is considered a strong point by some (the server doesn't have to worry about client starts and stops), in MVS a close is a critical operation. When a data set is closed, an explicit end of file mark is created, an assurance that the contents are intact. If there is no end of file, it can be assumed that the data set is corrupt.



The problem was handled by adding a close operation on timeout. Let's say a user does a lookup which results in the allocation (open) of a data set. Then, a timer is set. When the timer expires, the data set is closed. This means that if a user asks for the data several times, there may be several opens and closes (especially if the timeout window is short).

### *VM/CMS NFS File Server*

The implementation of NFS on the VM operating system provides some interesting perspective on the problem of mapping file attributes and names. As with MVS, a separate mount protocol is used to mount a CMS minidisk, including any translation services:

```
mount yktvmx:scanner.191
mount yktvmx:scanner.191,ro,pass=darkley
mount yktvmx:scanner.191,record=nl
```

In the first command, a minidisk is mounted. In the second example it is mounted in read-only mode and the password is specified. In the third example, translation is specified by adding new lines to the end of records.

The default translation is none, known as binary mode. This means data is simply read in a stream. New line mode adds a new line. In addition, text mode provides ASCII to EBCDIC translation.

Mapping of names is a bit tougher. There are three translations that the user can specify:

- Fold upper- and lowercase
- Allow mixed case with no translation
- Do file name conversion

A typical example of file name conversion is the case of long file names, which are not allowed in CMS. CMS also does not like special characters or names with two components. Figure 6-16 shows some example translations.

### Hardware NFS Implementations

Distributed file systems, diskless workstations, and the need for network-based archival and backup services have all fueled a ravenous appetite for enterprise-wide file servers. Several niche companies, such as Auspex and Epoch, were formed specifically to fill this need.

Auspex is a company founded specifically to provide a hardware solution to the Network File System. A typical NFS server is a general-purpose computer, such as a VAX or a Sun. General-purpose computers are not optimal for intensive data-access operations—bottlenecks appear that hinder the flow of data. Auspex engineers developed a computer architecture that

Unix Name	CMS Name
good.data	GOOD DATA
bletch.	BLETCH #N
bletch	BLETCH #N2
free_for_all.c	FREE_FOR_ALL@C#N
%free_for_all.c	FREE_FOR_ALL@#N2

## 6-16 VM and NFS File Translation

does nothing but provide disk storage services to clients, eliminating many of the bottlenecks in a traditional server. Figure 6-17 shows typical values for server configurations.

The Auspex architecture consists of a main processor and several intelligent peripherals, a term Auspex calls “functional multiprocessing.” The host processor runs Unix and is used for traditional housekeeping tasks, such as authentication at the beginning of a session. Most file requests, however, skip the host processor all together and go directly to specially configured boards.

Incoming requests are first received on the Ethernet processor. Up to four processors can be configured to support a total of eight different Ethernets. Each of the processors has the entire NFS protocol stack built in. Support for eight different Ethernets means that very large numbers of clients can be accommodated without swamping the bandwidth of a single Ethernet LAN.

Once a packet is received on the Ethernet processor, it moves over to the file processor. This dedicated board is responsible for mapping incoming requests from clients into the Auspex file system. Requests are satisfied from a main memory cache or moved over to the storage processor.

The storage processor is responsible for managing disk arrays of SCSI-based drives. Each rack of drives is an array of five 1002-Mbyte drives. Up to four racks are configured on a processor, with up to three processors per machine, allowing 60 Gbytes of secondary storage.

The Auspex architecture is ideal for diskless workstations (or workstations with only a small drive for disk swapping and a few key operating system files, known as a “dataless” workstation). A typical general-purpose server, such as a Sun-4, can support 10 to 20 active clients. Auspex systems support anywhere from 50 to 100 diskless or 100 to 200 dataless clients. For large departmental configurations, this is a cost-effective way to provide disk drives, while freeing up the general-purpose servers for other tasks.

Tier	Storage Type	Typical Values
Workstation	Main memory	8–64 Mbytes
	Disk drive	0–500 Mbytes
Compute Server	Main memory	8–128 Mbytes
	Disk drives	250 Mbytes–4 Gbytes
Auspex NS 5000	Main memory	34–114 (See Note)
	Magnetic disk	2–60 Gbytes
	8-mm tape	2.3 Gbytes
	4-mm tape	1.3–2 Gbytes
Epoch-1 InfiniteStorage Server	Main memory	8–20 Mbytes
	Magnetic disk	2.28–7 Gbytes
	Rewritable optical disk	30–60 Gbytes
	WORM optical disk	330–990 Gbytes
	8-mm tape	2.3 Gbyte/Tape
Note: Auspex has 18 Mbytes of fixed main memory on the multiprocessors, plus an additional 16–96 Mbytes available as a user-configurable primary memory data cache.		

## 6-17 Storage Hierarchy

### *Infinite Storage: Epoch*

Auspex uses the NFS protocols as a way of supporting large amounts of secondary storage. Another company, Epoch, took the same protocols to serve a different purpose: access to large amounts of tertiary storage for archiving and backup in enterprise networks.

Epoch systems combine the magnetic and optical disks into one transparent file system—a concept they call InfiniteStorage. A typical system has several different kinds of storage media: magnetic disk, 8-mm tape drives, rewritable optical disk, and write once/read many (WORM) drives. Typical storage capacities range from 30 Gbytes up to one terabyte (1000 Gbytes) of storage.

Using the NFS protocols, clients “mount” file systems on the Epoch servers onto their workstations. To the workstation, the files appear local. Think of this as giving each user one terabyte of on-line data—ideal for applications like computer-aided design or seismic analysis.

The Epoch operating system looks at how often files are accessed. If a file is not accessed for a while, it is moved off the magnetic disk onto the rewritable optical disk. If the file is not used there, it is moved onto either a tape drive or to a WORM optical disk. To the client the file is still available, but it slowly migrates onto slower (and cheaper) storage media.



The idea is to keep a file that is being used frequently on a fast medium, such as a magnetic disk system. If a file isn't being used, it can be put on a slower system. When the file is later accessed by a user, it is moved back to magnetic disk to provide faster access times. All the files are still available, but ones that aren't used just take a little bit longer to get to.

This movement of files from one medium to another is called staging. Typically once a day, a bulk periodic staging occurs. The system looks at all the files on magnetic disk to see if they should be moved.

Files are moved off magnetic disk until a low-water mark is reached: 75 percent utilization might be an administrator's choice for the low-water mark. The idea is to keep a balance. If too many files are moved off, they'll just have to be put back on later. If not enough files are moved, they'll have to be moved later when the disk fills up.

Periodically during the day, the drive may reach a high-water mark—say 90 percent utilization. If that occurs, the Epoch system uses an event-driven staging to move files and free up room. Based on size of file, latest access, and the administrator's preferences, enough files are moved to reach the low-water mark again.

The infinite storage concept works for both archiving and backup. The nice thing about doing a backup on optical drives is that the user is able to recover a file without intervention from the network manager. The network manager can choose which files should be archived—kept on-line—and which should be backed up, requiring user intervention to restore the file.

An extension of infinite storage is a new service on the Epoch systems which is called Renaissance. Renaissance adds the workstation into the storage hierarchy by dynamically moving data from client systems onto the Epoch system if it is not used. Once on the Epoch system, the data will migrate down the storage hierarchy until it is eventually placed on tape or a WORM drive.

### *A New Niche Market*

Auspex and Epoch both use the same protocols—NFS—as a way of providing specialized solutions in enterprisewide networks. Auspex provides very high-speed access to large amounts of secondary storage while Epoch specializes in massive amounts of tertiary storage.

The typical clients for a company like Auspex are organizations doing software design or CAD/CAM: environments requiring large files on workstations. Traditional servers quickly fill up and are saturated. Auspex customers, like Adobe Systems and LSI Logic, an integrated circuit manufacturer, use the servers as a way of satisfying the growing data appetite of powerful workstations. Of course, Adobe Systems and LSI Logic are also extensive Sun customers, mixing the Sun and Auspex equipment to form an integrated network.



Typical clients for the Epoch servers are oil companies and geological researchers, who need to keep very large amounts of seismic data available. An example is the University of California at San Diego which uses a 360-Gbyte Epoch system.

### For Further Reading

Chet Juszczak, "Improving the Performance and Correctness of an NFS Server," *Usenix Conference Proceedings*, Usenix Association, Berkeley, Ca., June 1990, pp. 53-63.

Bob Lyon and Russell Sandberg, "Breaking Through the NFS Performance Barrier," *Sun Tech Journal*, August 1989.

Rick Macklem, "Lessons Learned Tuning the 4.3BSD Reno Implementation of the NFS Protocol," *USENIX Conference Proceedings*, Usenix Association, Berkeley, Ca., January, 1991.

Bill Nowicki, "Transport Issues in the Network File System," *ACM SIGCOMM Computer Communication Review*, April, 1989.

Brian Pawlowski, Ron Hixon, Mark Stein, Joe Tumminaro, "Network Computing in the Unix and IBM Mainframe Environment", *Uniforum '89 Conference Proceedings*, March 1989.

R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, "Design and Implementation of the Sun Network Filesystem," *Usenix Conference Proceedings*, Usenix Association, Berkeley, Ca., Summer 1985.

Sun Microsystems, *Network Programming Guide*, Part Number: 800-3850-10, Revision A of 27 March, 1990.

———, *NFS: Network File System Protocol Specification*, RFC 1094.

———, *System & Network Administration*, Part Number: 800-3805-10, Revision A of 27 March, 1990. Contains information on NFS, NIS, and other general administration issues.



# RPC Tool





# RPC Tool

## RPC Compilers

Chapter 5 discussed how the Remote Procedure Call protocol allows a procedure to be moved to another system. RPC provides the glue that can transmit a request to a remote server for a procedure to be executed. To access an RPC service the user can write calls directly to the RPC library. Writing low-level RPC calls kind of defeats the purpose of remote procedure calls, which is the ability to view remote computers as an extension of the current system with a transparent network interface.

The direct approach to using the RPC library requires the programmer to treat a foreign system differently from the local system. For local calls, a simple function call is executed. For remote systems, the RPC request must be built and then decoded.

An RPC compiler allows the programmer to deal with a higher-level language which allows, to a great extent, all procedures to be treated as if they were local. Procedures are declared as if they were a local procedure, being passed parameters and returning a variety of pieces of data (see Fig. 7-1).

There are two different RPC compilers available for the Sun RPC protocol, the original `rpcgen` and a more recent addition, the Netwise RPC Tool. `Rpcgen` has the advantage of being widely available in the freely-licensed RPC source distribution and forms part of the NFS reference implementations, but it is based on older technology than the Netwise software.

The Netwise RPC Tool is a commercial system which makes the RPC protocol easier to use and adds value in the form of better error checking, binding, and other higher-level functions. Netwise defines a specification language to use as input; that language is very similar to the C declarations that are used for defining a local program. This specification file and the original code for each of the various procedures in an application are bundled together with code generated by the compiler and libraries that come with the system.

To the programmer, the network interface to a distributed program looks quite similar to undistributed programs. It is possible through a variety of customization mechanisms to go beyond this transparency and begin con-

trolling the RPC mechanism. Programming in a distributed environment can be transparent, because the underlying compiler takes care of many of the issues that arise when a program is distributed. Several issues complicate separating the two procedures on different machines:

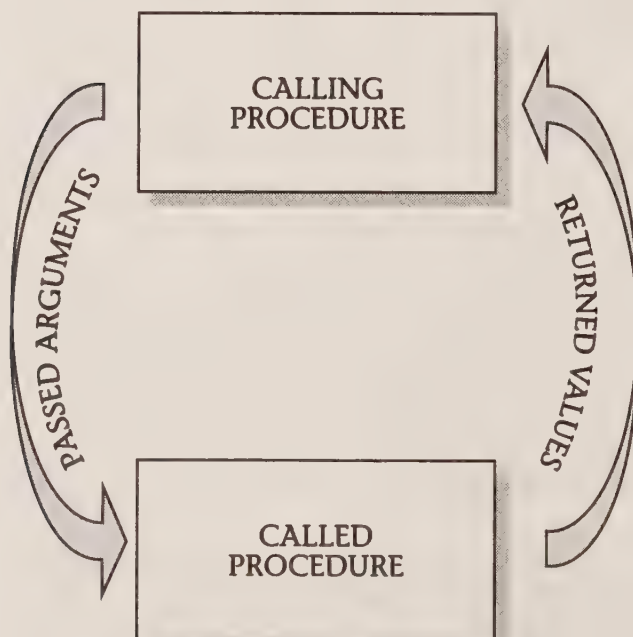
- Locating and forming an association (binding) with the remote procedure
- Representing the data in a common format
- Sending and receiving the data
- Handling errors that do not exist in a single-machine environment

A distributed environment can lead to errors that would not happen on a single machine. A distributed application often requires several steps to occur on each machine compared to a single action in a non-distributed application. In a distributed environment, each machine will fail independently. The distributed application must be able to decide the effect of a single component failure on the integrity of a transaction.

Single-machine communication between procedures is a special case of interprocess communication (IPC). When the applications are distributed, RPCs provide an extension of the IPC mechanism into a multimachine environment.

### Netwise RPC Tool

Netwise makes a compiler, the RPC Tool, that converts a series of procedures that run on one machine into a series of procedures that run on several machines. The compiler handles the issues of how the remote pro-



cedure is to be activated and how to pass and return complex data structures, as well as various miscellaneous issues such as the use of pointers and handling errors. The compiler provides a syntax to the programmer to use for registering, locating, and binding procedures.

The original, single-machine application consists of a series of user procedures. An application programmer distributing an application defines an interface specification which is processed by the RPC Tool to yield four additional types of procedures:

- Client and server stubs
- A dispatcher procedure
- Pack and unpack procedures
- Network and support libraries

These additional procedures are linked with the original application procedures to form the two components of the distributed application: the client and server programs.

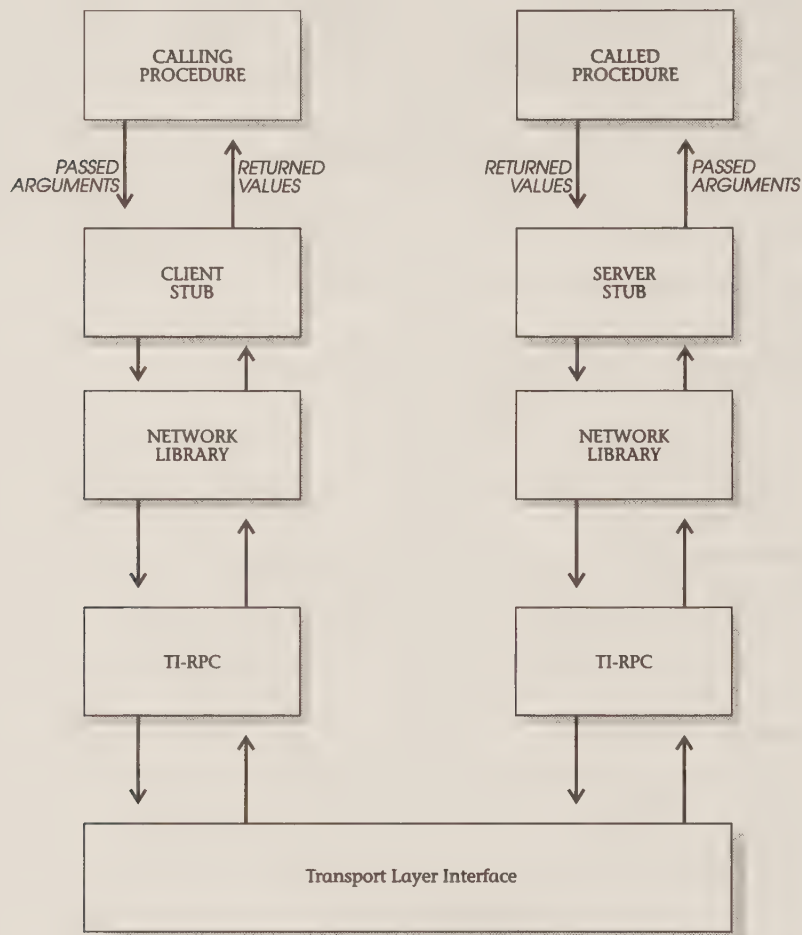
The key elements from the programmer's point of view are the network library and the client and server stubs (see Fig. 7-2). Because many of the mechanisms in the RPC Tool operate by default (a programmer can simply provide a list of which procedures will be distributed), it is possible for programmers to be totally unaware of the distributed environment—the network is treated as if it were simply one large distributed machine. For complex applications, the programmer can modify each of the elements in the RPC Tool to provide better control or improve performance.

The client and server stubs emulate the single-machine interprocess communication mechanism. The client stub looks like the original called procedure to the client application. It accepts the same parameters as the original procedure and will return the same values. After accepting parameters, the client stub delivers the request for a procedure call to the network library. The network library is responsible for converting the arguments to canonical network representation, serializing the request in preparation for transmission, and delivering the request to the remote server. The network library does so by mapping the RPC Tool stub calls into transport layer calls, thereby providing the program independence from any particular transport layer.

The transparency of the code produced by the RPC Tool is a very important attribute. The programmer designs procedures without worrying about which ones will end up on which servers. Especially important is the fact that the underlying transport mechanism of the network is also transparent to the programmer.

Once the data is delivered to the remote machine, the message is delivered to the server stub, which invokes the original user procedure. The client and server stubs act as a network glue to join the original client main body to the distributed procedure. The server stub calls the remote procedure, passing in the appropriate arguments. The server stub then takes the return values and packages them into a message, which it delivers to the





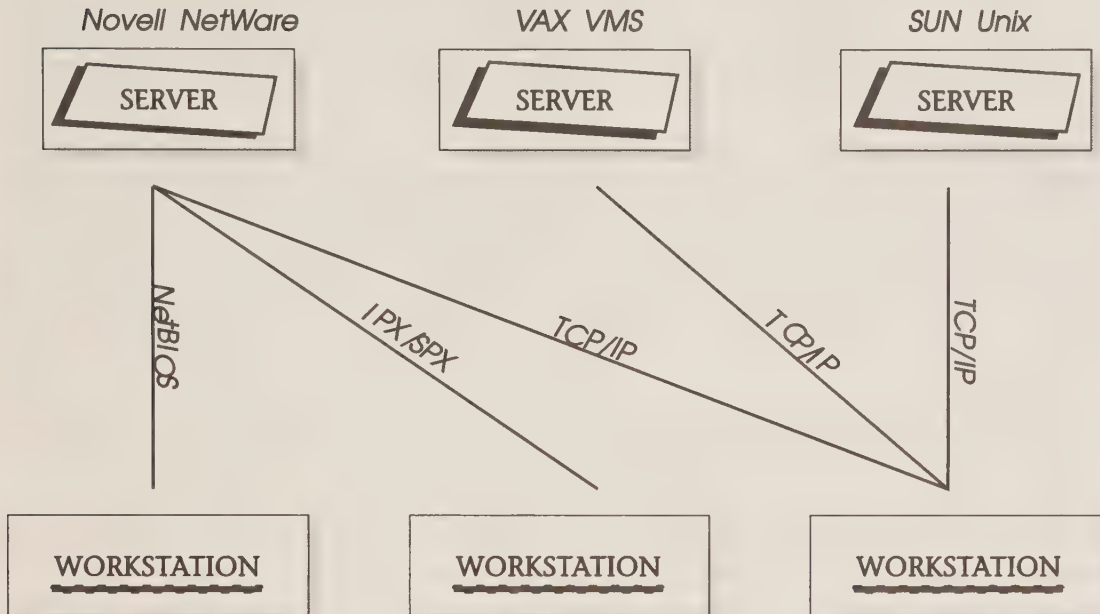
## 7-2 Remote Procedure Call

network library. Once again, the network library is responsible for delivering the resulting data back to the client. There, it hands the reply to the client stub. The client stub takes the reply and packages it in the way that the calling procedure expects. It then returns to the calling procedure, just as if the local IPC mechanism had been used.

The network library allows the selection of different transport mechanisms at runtime (see Fig. 7-3). By feeding the specification file into a different version of the RPC Tool, it is even possible to support different RPC mechanisms. Writing to the RPC specification file means that a program can be written without knowing the type of network it will run in.

Assuming the program uses a standard C syntax, application code can quickly move from one machine to another. A program that makes conservative use of system resources could be written for both the Novell Network Loadable Module and for a SunOS Unix server. Issues at the transport layer, like IPX/SPX in the Novell world or TCP/IP in the rest of the world,





### 7-3 Heterogeneous Environment

are totally transparent to the programmer. Transport layers are shielded by the RPC protocol, which in turn is shielded by the RPC compiler and the RPC Tool.

The RPC Tool is certainly not the only approach to the problem of writing portable, distributed applications, but it does have some significant advantages in the environments discussed in this book. The RPC Tool has been adopted by Novell as the RPC mechanism to use on its networks. What makes the RPC Tool especially attractive is that Netwise also supports a variety of other platforms. The computers supported by Netwise include all of the 80x86 computers, VAX, Sun, IBM, and many other machines. Supported operating systems include MS-DOS, Microsoft Windows, OS/2, DEC's VMS and Ultrix, and Unix.

The types of networks supported include NetWare, NetBIOS, TCP/IP, DECnet, and IBM's SNA/APPC. An IBM mainframe, for example, would probably use the SNA network protocols. A Novell workstation could communicate with a program on the IBM using the SNA gateway and the RPC Tool. Similarly, NetBIOS could be used to allow 3Com 3+ workstations to communicate with a Novell workstation that has the NetBIOS shell.

### Procedure Declarations

The RPC compiler takes as input an RPC specification. This specification is a series of procedure declarations. The programmer prepares one entry for

every procedure in an application that will be run on a remote machine. The RPC specification is then used as input for the RPC compiler. The compiler takes the specification and generates a series of procedures which will be linked with the original program: client and server stubs and pack and unpack procedures.

All data shipped from one machine to another has to be packed and unpacked. Packing data means translating it into a machine-independent format. In the case of Netwise, the formats supported are XDR and the ISO standard known as ASN.1 BER. Unpacking means translating it back into a machine-dependent format. The client and server stubs call the pack and unpack actions whenever a message is about to be sent or has been received.

The specification can be very simple, consisting basically of just a list of the procedures that will be accessed remotely. It is also possible to provide custom procedures, such as error recovery, in the specification.

When the programmer defines a remote procedure, the programmer includes the definition of any passed arguments and returned values that the procedure will use. The programmer can also declare that certain variables are global—accessible to several different procedures without having to pass them explicitly. The RPC Tool supports global variables because many existing applications make heavy use of this technique as part of their interface. Global variable support in the RPC Tool makes the task of porting existing applications to a distributed environment easier.

The definition of data structures uses the same method as the C programming language (which is the language used in the code generated by the RPC compiler). Simple data types, such as integers or characters, can be defined. Simple data types can be combined to form complex data structures.

To illustrate the concept of data structures, take a hypothetical program that operates as a database server. It will need to receive requests from clients for data. The programmer defines a structure called `DataRequest` with several elements. The first element is the command type. In the case of the database server, this signifies whether the operation is a read, update, delete, or append of data to the table. This first element will probably be defined as an integer. The second part of the structure might be the SQL statement itself, which is defined as a string of characters.

For most data types, the programmer uses the same syntax in the RPC Tool that would be used in the ANSI standard for the C programming language. Two exceptions are the complex data types of arrays and unions.

An array is a set of one simple data type. An array of characters, for example, would hold all the individual characters of a text string. A database is well suited to an array. Each line of a table in a database can be considered an entry in an array. The entries combined make up the full array.

The problem with arrays is that often not all of the positions in the array are filled for a particular request. An array might be designed to hold 300

rows of data but a particular request might only retrieve 12 rows. In a single-machine environment, unused space in an array is not a problem (assuming there is enough memory). But, over the network, transmitting empty cells can dramatically affect the amount of network traffic produced by any one request.

Netwise handles this problem by allowing the programmer to specify a “termination expression” for the array, in effect implementing variable length arrays. After the last unused cell, the programmer would insert the termination value (for example a NULL character) into the next cell. When the pack procedure packs the data, it will recognize the termination value and stop packing.

The second data structure is the union. A union consists of several sets of elements, only one of which can be active at any one time. Again, in a single-machine environment, the only problem with a union is the use of memory space. In a networked environment, sending all of the members of a union through the network, even though only one is being used, is a waste of network bandwidth, memory, and CPU cycles.

To supplement the union definition, the RPC specification includes a choice expression, which is executed at runtime to determine which of the variants of the union is active at that time. Only the active variant is sent over the network to the remote procedure.

## Process Binding

Binding specifies how the relationship between a remote procedure and the calling program will be established. A binding is formed when two applications have made a connection and are prepared to exchange commands.

If a procedure is called rarely, a nonpersistent style of binding is used. Nonpersistent binding means that a connection is set up on the underlying transport mechanism when the remote procedure is called. As soon as the values are returned, the connection is dismantled. Since connections on a network are a limited resource, the nonpersistent style is used to conserve those resources. Rather than keep a connection in place for infrequent communications, the connection is only set up when needed.

Nonpersistent binding means that each call has to set up and tear down a separate connection. The additional overhead involved in connection establishment makes nonpersistent binding inappropriate for remote procedures that are called frequently by the same caller.

For applications that make many repeated calls to remote procedures, a persistent style is used that allows several different calls to all use the same connection. The connection is set up and then kept open. The connection ID is kept in memory, and that connection is used for subsequent calls. When the application decides that it no longer needs remote resources, the last procedure call tells the server that the connection is about to be dismantled. The opening and closing of the connection are usually kept in the initialization and termination procedures of a program.



For both types of bindings, the address of the remote server can be determined at runtime. The address can be specified as a name or internetwork address. If the remote computer where the server will run is always the same, an internetwork address can be used to specify the target. Of course, it is hard to predict the future, and it is increasingly common for applications to move to different machines because the application is popular with users and a larger computer is needed to handle the increased demand.

Increased flexibility in moving procedures is available by using names. At runtime, the name of a server is specified, such as DatabaseServer. The network library uses the appropriate mechanism to translate that name into an internetwork address.

In a Novell environment, for example, the RPC network library would use the Service Advertisement Protocol (SAP). In a NetBIOS environment, the network library would use the name management commands of NetBIOS to find the location of the designated target. In a Sun environment, NIS, NIS+, or the Domain Name System would be used.

Using abstract names for a service allows the network manager to move a program from one computer to another without changing the program on either the client or the server. When the location of a server changes, the initialization procedure in the server control program for the application will inform the name service (e.g., the NetWare bindery) of its new location. Workstations requesting the remote services will be informed by the name service of the current location of the service. The use of naming will be discussed in Chapter 9.

## The Client Stub

To see how the code generated by the RPC Tool works, it is useful to begin with the client stub. The client stub is responsible for emulating the remote procedure—for accepting the passed arguments and then providing the appropriate return values.

The client stub is a piece of code that is structured as a state machine. A state machine is a procedure which goes through a series of states, one after the other. During each state it executes a certain set of instructions and then, if the appropriate conditions are met, proceeds to the next state.

First, the client stub enters the start state in the state machine. The start state initializes any data structures used in the client stub. If necessary, the start state also initializes the network library.

Second, if the binding is based on a network name, the machine enters the find state, which maps the server name to a network address, using a support procedure in the network library. The network library uses whatever mechanism is necessary to perform the translation on that particular network.

Third, the machine enters the connection state. This establishes a connection with the remote server and provides the client state machine with a



connection ID, which might be used for future calls if this is a persistent binding.

Fourth, the pack state is entered. Here, the client stub state machine calls pack procedures that were generated by the RPC compiler. The pack procedure returns a message known as a protocol data unit (PDU). The PDU is the equivalent of a sealed envelope in a messaging system and is ready to be submitted to the transport layer and transmitted over the network.

Fifth, the machine enters the send state, which hands the PDU off to the network library to be sent over the network. The client machine waits for the reply to be received and then enters the get state. When the reply is received, the client machine unpacks the header of the incoming PDU to make sure that it is the answer to the request it sent and not some stray that somehow reached this socket. The client machine then checks the reply to see if an error occurred during processing. If not, it unpacks the message and places it in the appropriate data structures.

Finally, if this is a nonpersistent binding, or the last call of a persistent binding, it enters the close state, which terminates the binding with the remote procedure. In the finish state, it stops execution and returns control to the calling procedure.

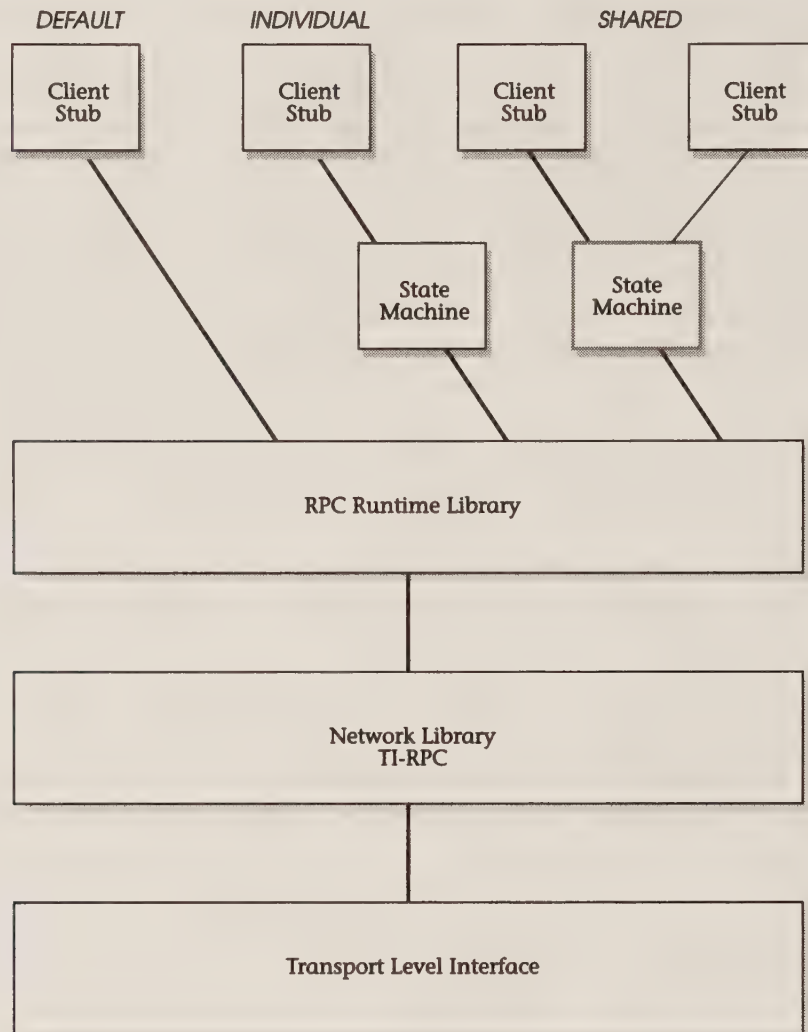
Many of the operations in these state machines can be shared among multiple procedures. Although the pack and unpack procedures are different, much of the remaining code can be reused. The RPC compiler allows state machines to be shared among different client stubs, thus reducing the amount of memory used at runtime. In fact, if the programmer uses the defaults provided by the RPC Tool and does no customization of the RPC specification, the code uses a default state machine that is built into the network library support routines (see Fig. 7-4).

## Server Control Procedure

There are several components on the server that are used to coordinate access to the various remote procedures. All of these components operate under the control of the server control procedure (see Fig. 7-5).

The server control procedure (SCP) accepts incoming messages and determines if they are bind requests for initializing a connection or procedures that are to be executed. If the incoming request is a procedure to be executed, the server control procedure hands the message to the dispatcher procedure. The dispatcher unpacks the header of the message to determine which procedure is to be executed and then hands over control to the appropriate server stub. The server stub unpacks the body of the request and places the data into the appropriate data structures. It then calls the remote procedure and waits for the return values.

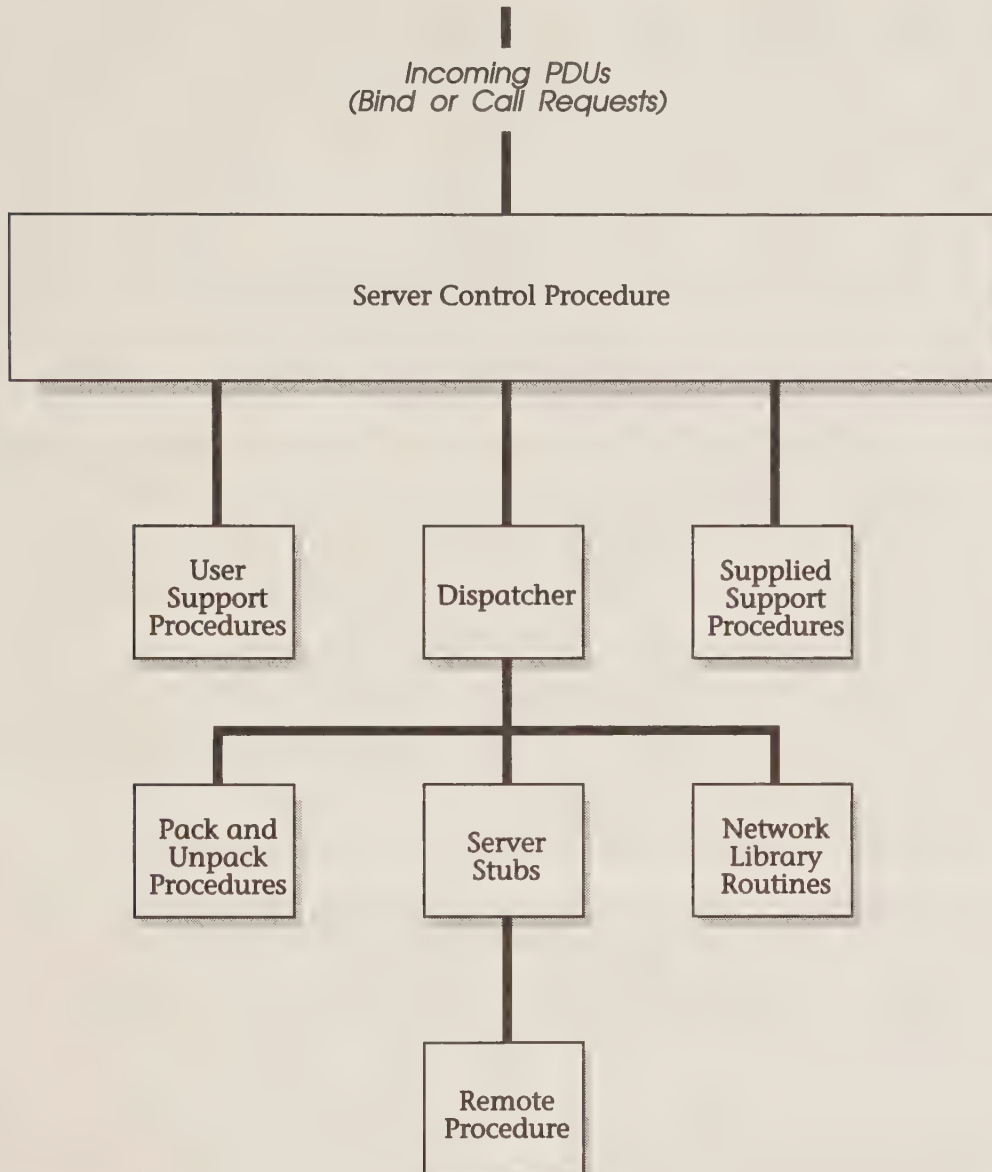
The return values are placed into a reply message and submitted to the network library, which then sends the reply to the client stub. The client stub in turn hands the return values back to the calling procedure.



7-4 Shared State Machines

The RPC Tool includes three types of server control procedures. Each is suited to a different type of server:

- The single-binding control procedure can accept a single bind request at a time. It then executes the requests one by one (single tasking).
- The multiple-client control procedure is able to manage several different connections at once. It takes each incoming message and determines which dispatcher to use. Note that this is still a single-tasking solution—only one request is serviced at a time. If other requests come in, they are queued.
- The multitasking control procedure is able to have several active procedures over different connections. The multitasking nature of this control procedure only works on those computers that have a multitasking operating system.



### 7-5 Server Control Procedure

Related to the multitasking control procedure is the multithreaded server control procedure. Unix is an example of an operating system that provides multitasking. For Unix, the programmer would use the multitasking SCP. For OS/2, the programmer would use the multithreaded SCP.

The server control procedure goes through several steps. First, it registers the name of the server using the network library. Second, it is responsible for managing all process bindings. Third, it is responsible for calling the dispatcher for routing incoming client requests. Finally, the server is also responsible for managing shutdown of the server, including freeing all memory resources and terminating all active connections. As with other

portions of the RPC code, it is possible to customize the server control procedure.

### Custom Procedures

There are several mechanisms in the RPC Tool that allow the programmer to customize its operation. The customization might be used, for example, for debugging or providing more sophisticated error recovery procedures. At the extreme, it is possible for the programmer to modify the operation of the state machines or even use the network library directly. Customization of the RPC code is done using four mechanisms:

- “Entry procedures” are executed whenever a state machine is entered. An entry procedure for a client stub is executed every time that the procedure is called by the calling application.
- “Exit procedures” are executed whenever the state machine is finished. In the case of the client stub, the exit procedure is executed whenever the reply is sent back to the calling procedure.
- “Hooks” are executed whenever a different state is entered within a state machine. The hooks are activated, for example, after a machine name is translated into an internetwork address (the *find* state) and before an attempt is made to set up a connection.
- “Traps” are executed whenever an exception condition occurs, which is usually when an error occurs.

A common customization is different error handling. For example, the programmer might put in a trap that looks for special errors caused by the network. The programmer would have the operation retried up to a retry limit instead of immediately returning to the calling procedure with an error.

Another type of error handling is to handle application-specific errors. The server stub state machine would have a trap that looked at some portion of memory to determine what type of error occurred. For example, the database server might have refused to process an update because the user did not have the security level necessary to do so.

The RPC Tool provides a series of macros (utility programs) to help the programmer determine what is occurring within the state machines. For example, the user can check the current state of the machine and even control the flow from one state to another.

Customization is available in several places in the control procedure on the server. First, the server control procedure will always execute any user-supplied initialization and termination procedures. For example, the user might log activation of the server upon initialization. When the server is shut down, a user-supplied termination procedure would log that fact and calculate the amount of time that the server was active.

It is also possible to provide more sophisticated control over the server using a server loop procedure, which is executed while the server is waiting



for an incoming bind or remote procedure request. A user-supplied server loop procedure might be used to determine if the number of active sessions is approaching the capacity of this server. If so, another server could be initialized, possibly on another machine.

The most fundamental level of customization is to modify or bypass most of the built-in mechanisms, such as the dispatcher and the stubs. One possible use would be to change the nature of the RPC calls from synchronous to asynchronous.

Under the RPC model, when a calling procedure wants to use a remote procedure, it stops processing until the result is returned. This is synchronous processing—steps occur in a predetermined order—the model under which most RPC mechanisms occur. Of course, in a multithreaded environment, the master application can keep processing because it starts a new thread that waits for the result of the remote procedure.

Asynchronous processing allows the calling procedure to continue working while the remote request is being processed. A simple asynchronous model might modify the client state machine to return immediately after the request is sent out.

Few programmers would want to make such fundamental changes to something like the RPC Tool. Rather than make such drastic changes, another paradigm of network programming such as NeWS or the X Windows System would be investigated.

## ISO and Sun RPC Mechanisms

Version 2.0 of the Netwise RPC Tool is based on ISO standard 8825 for the encoding of data, known as the Abstract Syntax Notation One (ASN.1) Basic Encoding Rules (BER). Built underneath that are Netwise mechanisms for providing session layer support. An example is the dispatcher that takes incoming messages and routes them to the appropriate server stub state machine.

Version 3.0 of the Netwise RPC Tool supports both the Netwise model for RPC mechanisms and the Sun Open Network Computing (ONC) model (see Fig. 7-6). The choice of the mechanism is decided at compile time and is transparent to the programmer. With TI-RPC, the transport layer can be selected at runtime. With TI-RPC, the programmer can take advantage of TCP and OSI-based protocol stacks, allowing the application to choose the appropriate path at runtime.

## Other Interfaces to the Network

The RPC Tool is a way for a distributed program to be developed with fairly sophisticated interaction between clients and servers. There are some types of operations, however, that are best handled by other styles of programming. In particular, control of the user interface in a networked environment is best handled by a Graphical User Interface (GUI) standard, such as

the X Windows System or Sun's Network Extensible Window System (NeWS).

The X Windows System was developed at MIT and is widely available on Unix, DEC, and IBM workstations. It allows the computer with the screen to be separated from the application program. The X Windows System is based on a bit-model of the workstation screen and provides a series of rules that govern which application can write to which portion of the workstation screen.

The advantage of a mechanism such as the X Windows System is that a cheap workstation can be purchased. The workstation can be nothing more than a dedicated X Windows terminal. These devices run nothing but an X Windows System server and depend on other servers on the network to run the applications.

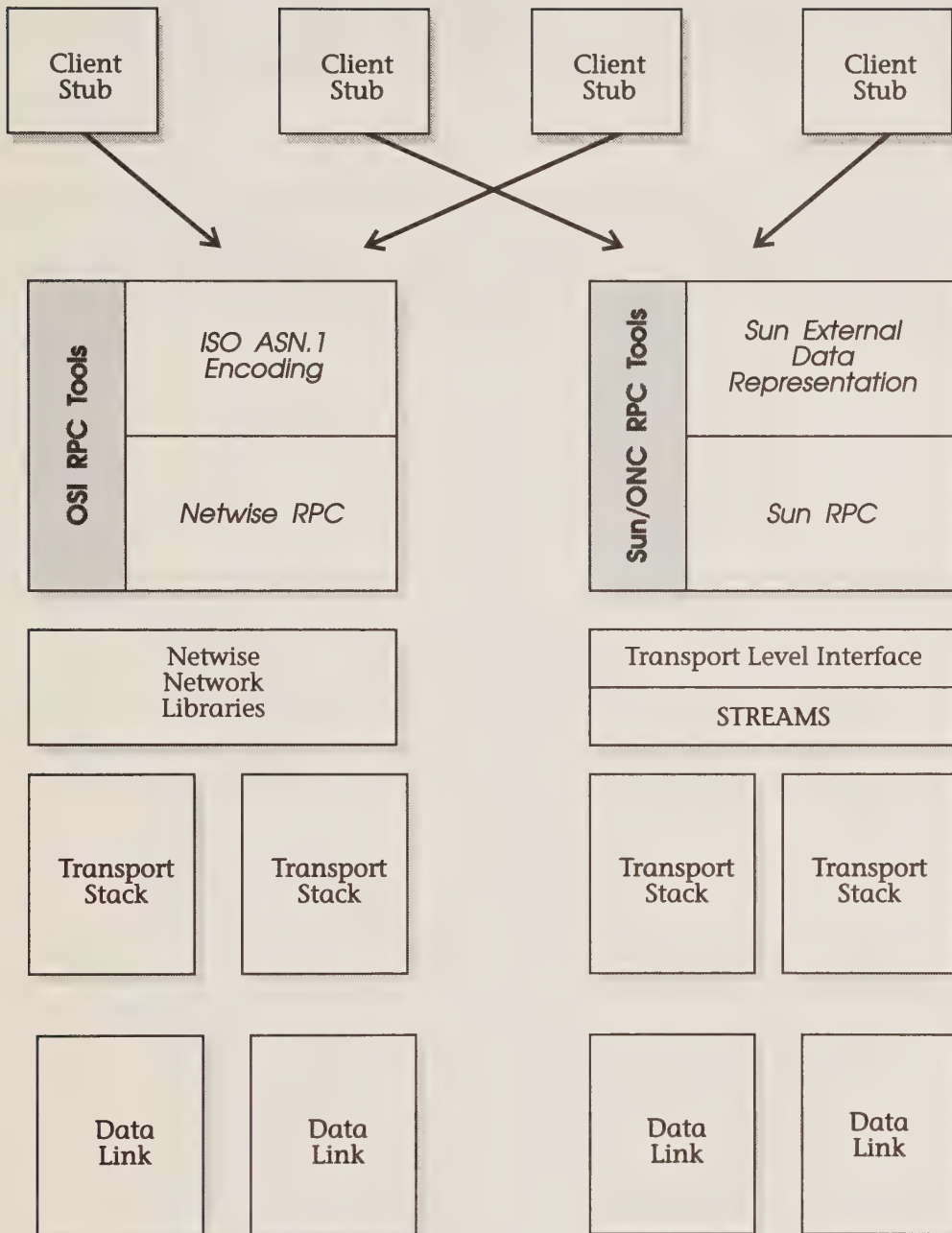
The X Windows System is a widely available method of allowing different brands of servers to all communicate with a user's graphics screen, mouse, and other devices. These application programs would then use RPC to communicate among themselves. For example, take a mail system. The mail tool, the user's graphical interface, would use a mechanism such as the X Windows System to communicate with the user's screen, performing functions like making the flag on the mailbox go up when mail arrives.

The mail tool program would then use RPC to communicate with mailers and directory services. It would probably use NFS so that the files containing mail could be located elsewhere.

The X Windows System and the RPC protocols allow a very wide distribution of work on the network. Specialized servers can be purchased and the configuration can be matched precisely to the type of service they will provide. Disk servers can have large amounts of disk but very few asynchronous ports, whereas print servers can be configured with a large spool space and many printer ports.

The X Windows System is a good standard for the control of windows, but it lacks something as an imaging model, which is based on painting lots of bits. Display PostScript is often added to the X Windows System to provide a language that works inside a window. Sun adopted this approach with NeWS, as did DEC with DECwindows. Other imaging models can also be used, such as the Phigs 3D graphics libraries. Built on top of the windowing systems are libraries that provide a common look and feel for all applications. This allows things like menu buttons to act the same way, look the same, and provide an easier way to use the user interface.

These look-and-feel standards, such as Open Look (used on Sun and many other workstations) or Motif, developed by OSF, are usually made available to the programmer in the form of a toolkit. The application programmer thus sees two sets of network interfaces: an RPC specification language and a GUI toolkit. In addition, the programmer should probably know about naming, security, and file access (NFS) so that intelligent decisions are made about how to construct programs.



## 7-6 Alternate RPC Mechanisms

In a Sun network, the windowing environment is known as OpenWindows. OpenWindows is actually two window systems: X and NeWS. Various programming interfaces and toolkits are available, such as APIs for X and NeWS. Because NeWS implements on PostScript, the programmer has the full power of this display language available in developing applications. Combined with the RPC Tool and TI-RPC, the programmer has all the tools available to develop sophisticated, distributed applications.

#### For Further Reading

Sun Microsystems, *ONC RPC Application Toolkit for SunOS Installation and Operation Manual*, Part No: 800-4815-10, Revision A, April 1991.

———, *ONC RPC Application Toolkit Transport-Independent RPC Reference Manual*, Part No: 800-4813-10, Revision A, April 1991.



# Network Time Protocol



# Network Time Protocol

## Why Bother?

In a study of hosts on the Internet that provided time services to other hosts, David Mills of the University of Delaware was able to find a total of 8455 hosts that provided an apparently valid time indication. The accuracy of the time returned varied from 11 milliseconds to over 11 days.

Why is time important? In a network there are a variety of services that depend on synchronized clocks. For example, Secure RPC uses a verifier that allows two hosts to make sure that some other host has not imposed itself in the middle of the conversation. The verifier consists of the time encrypted with a shared DES key.

In the initial exchange between hosts using Secure RPC, a window is agreed upon. The window is some period of time from the present to some offset into the future. If a verifier is received that contains an encrypted timestamp outside the window, it is rejected. The purpose of the encrypted timestamp with a window limit is to prevent replay: having a third host keep track of a conversation and then replaying it at a later time as a way of obtaining data.

If the clocks are out of synchronization between two hosts, the window mechanism does not work properly. Timestamps will be received that are outside the window, causing rejection of the verifier. The hosts will then have to go through the process of establishing a session again (which will quickly be rejected because clocks are out of sync).

Valid time is also important in an environment with distributed database updates. If two remote clients both want to update data or obtain locks, we usually determine the order of the updates or the receiver of the locks based on the time of the request. Again, synchronized clocks are essential.

There are several means that a host can use to obtain the correct time. Unfortunately, on many computers this method consists of the wristwatch method: the systems programmer looks at a watch and uses that to set the

time. Even if the time is properly set, this does not mean that the clock remains accurate. Many clocks are maintained by some form of battery-backed device that may drift up to a second per day (and is rarely corrected).

We can already see two potential sources of error between clocks: they can be initially set differently and they can drift (at different rates) over time. Once time becomes an important synchronization method, it is essential to provide a method to allow different hosts to coordinate their clocks and keep them coordinated.

In the TCP/IP suite, there are two fairly primitive protocols that can be used. The ICMP timestamp option is one (although the ICMP timestamp does not give you the date with the time); a second is the TIME protocol, documented in RFC 868. The TIME protocol is quite simple: a time server listens on UDP or TCP port 37. When it receives a request, it sends the time. Time is sent as a 32-bit number which is the number of seconds since midnight on January 1, 1900 UTC. The ICMP timestamp option is documented in RFC 781 and is basically similar to the TIME protocol.

## NTP

A newer protocol is the Network Time Protocol (NTP), which is documented in a series of RFCs and journal articles (see “For Further Reading” at the end of this chapter). A much more sophisticated protocol than ICMP or TIME, it allows coordination of clocks in a large network with specific provision for very fast networks (which require very precise clocks).

NTP serves two basic functions; it is:

- A way to coordinate the distribution of time from a set of servers to a client
- A way to synchronize the clock frequencies

Synchronizing frequency means clocks in the network run at the same frequency: they advance at the same rate. Synchronizing time means that the clocks all agree upon a particular epoch with respect to Coordinated Universal Time (UTC). To synchronize clocks means making them agree on both frequency and time.

What makes this difficult is the variable delay between two computers in the Internet. In a local-area environment, it is fairly easy, over the course of a few measurements, to synchronize time within a few tens of milliseconds. The coordination of clocks over a wide-area environment may require many measurements over many days.

Time is ultimately derived from International Atomic Time (TAI), which is accurate to a few parts per trillion. Because the mean solar rotation time



of the Earth is slowly decreasing (at least now), there is another form of time, Coordinated Universal Time (UTC), which is presently decreasing a fraction of a second per year relative to TAI—hence the introduction of a leap second into the calendar between 1990 and 1991.

UTC is known as the primary reference source. In the Network Time Protocol, a primary server is one that is connected to this primary reference source by wire, radio, or satellite. In addition, there are secondary servers which derive time from other secondary servers and from primary servers (see Fig. 8-1).

The primary and secondary servers are arranged in a hierarchy. We assume that there is a decreasing level of accuracy as we go down the hierarchy. Each level of the hierarchy is given a number, called a stratum. The root primary server has a stratum of 1. A secondary server that gets its time from the root has a stratum of 2. The stratum is important because the farther down the tree we get, the greater the potential of an unreliable time server.

### *Message Exchange*

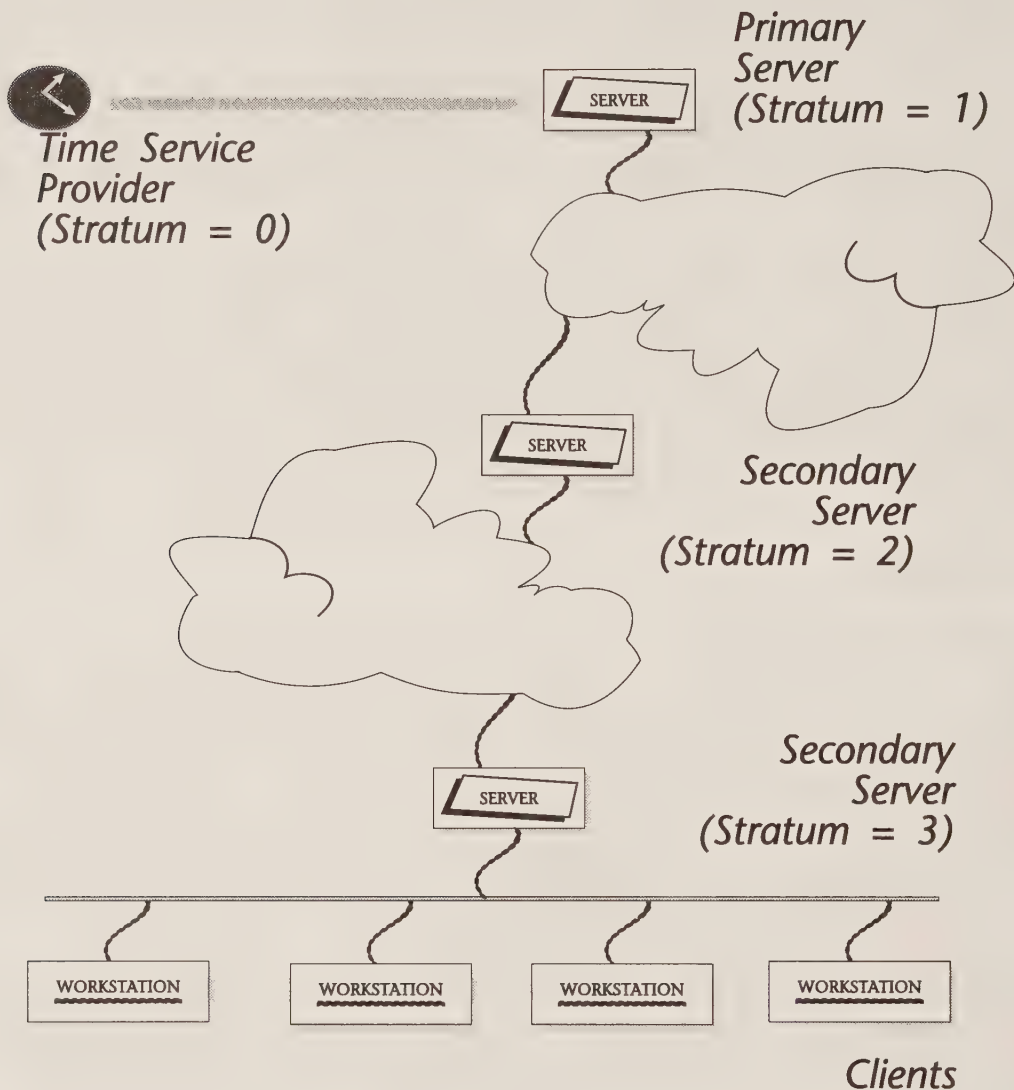
The basic process is for a client to send an NTP message to one or more servers. The server takes the incoming datagram and interchanges the addresses, overwrites a few fields in the message, recalculates the checksum, and sends it back. The result of this process is four timestamps for each message exchange (see Fig. 8-2).

Given a set of four timestamps, a server can calculate two quantities: the delay and the offset (see Figs. 8-3 and 8-4). The delay is how long a packet spent on the network, a quantity that can only be estimated since the clocks that report on these events are not synchronized.

In a local-area network, the round-trip delay is a small fraction of the clock offsets and can often be discarded. In a wide-area environment, however, the delay can easily be more than the offsets, and since the delay may have considerable variance, further processing of the data is needed.

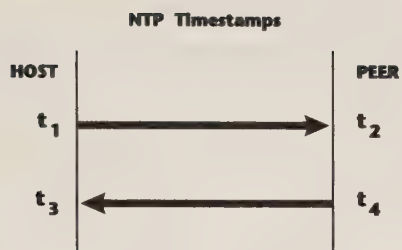
There are a variety of methods to decide which clock offsets are the correct ones, based on collecting several pieces of information from different servers. The round-trip delay and the clock offset for a given peer are calculated from the four most recent timestamps. Over time, a set of delay and offset estimates is accumulated. The problem is that given a set of these numbers, we must determine the true offset and delay (or some relatively accurate estimate of the truth).

Delays are typically increased because the source of the estimate is far away or the network becomes congested. Studies show that as the delay increases, the estimates of the clock offset also increase. This makes sense, since the packet spends longer in the network and thus the clock offset number is artificially increased.



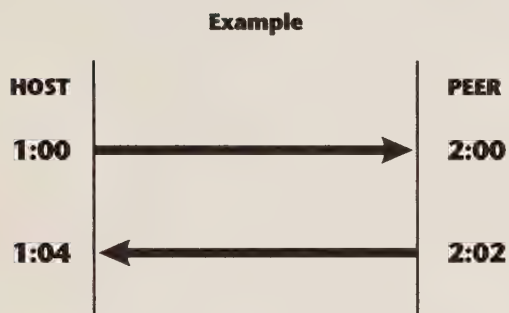
### 8-1 NTP Network Configuration

To take out stragglers, NTP uses a filter that sorts arriving estimates in order of increasing delays. Those with the lower delays are considered the better estimates. Again, a register with eight slots is used. The register holds offset and delay samples, and in Version 3 of NTP, an estimate of dispersion. We start by taking the first eight estimate pairs and adding



**8-2**  
NTP Timestamps

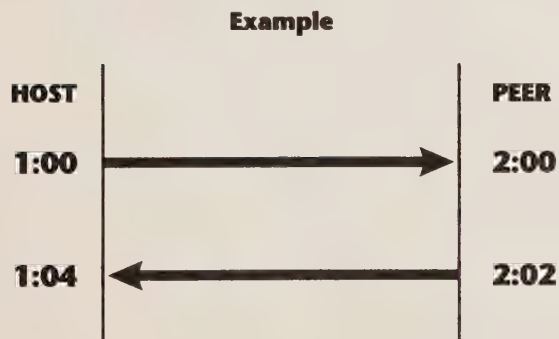
$$\text{offset} = \frac{(t_4 - t_3) + (t_1 - t_2)}{2}$$



$$\text{offset} = \frac{(-58) + (-60)}{2} = -59$$

**8-3**  
Calculating the Offset

$$\text{delay} = (t_4 - t_1) - (t_3 - t_2)$$



$$\text{delay} = (4) - (2) = 2$$

**8-4**  
Calculating the Delay

them to the register. When the new estimate comes in, it is added to the rightmost position of the register, pushing off the leftmost member.

Once we have a set of delay and offset estimates for a given peer, the next issue is how to determine which peers to trust. As a general rule, we can assume that most time providers are true and that the number of “false tickers” is relatively small. Another assumption we can make is that most true time providers provide time closely clustered around the real UTC.

We can make two more assumptions. First, a provider with low stratum number—closest to the root—is more likely to provide good time estimates. Finally, providers with low dispersion—a low variance in their estimates—tend to be more accurate.

The question now is, given a set of peers, each with a delay and offset estimate and a dispersion of their estimates, which one is to be considered the true source to be used in updating the local clock?

NTP takes the list of active peers and sorts them by stratum first, then by dispersion. Before a peer can be included on the list it must first pass some basic sanity checks. For example, the source of the time for the peer should not be the host itself or a synchronization loop will have been introduced.

Several other sanity checks are used. The dispersion number has to be based on a register of samples that is half full: otherwise the dispersion is based on a very low sampling rate. This prevents the use of data from broken (or very new) associations. If no peers pass the sanity checks, the local clock just keeps running at its natural pace.

Once the list has been pruned with sanity checks, it is set to a maximum size (currently five). Then, the list is once again truncated as soon as the number of strata is more than two. This means that the candidate list includes no more than two different levels of the time hierarchy.

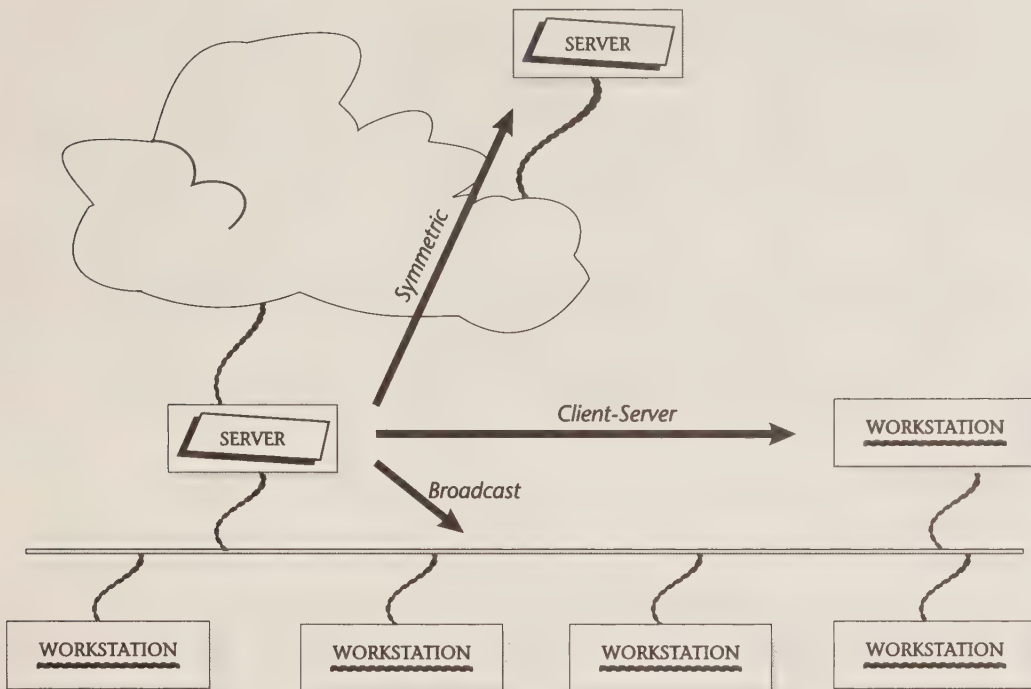
Finally, the list is sorted again. This time it is sorted first by stratum and then by the distance between the host and the peer. Then, the peer with the maximum dispersion with respect to the remaining peers is thrown out and the procedure is repeated until a single peer remains. That peer becomes the clock source.

## Associations

For a particular association between two computers, a computer can be in one of five modes (see Fig. 8-5):

- Symmetric active
- Symmetric passive
- Client
- Server
- Broadcast





### 8-5 NTP Relationships

The two symmetric modes are used for a scenario in which two peers can become the reference source for each other. These two modes are an indication that a host is willing to both synchronize and be synchronized by a peer. The active mode is for two peers that are near the bottom of the hierarchy and will thus be frequently exchanging messages. The passive mode is for nodes near the top of the hierarchy—with low stratum numbers—that will be infrequently exchanging messages. The passive peers await requests, while the active peers have preconfigured tables that point them to passive peers further toward the primary root.

The broadcast mode is for high-speed LANs with many workstations. In broadcast mode, one or more servers periodically broadcast messages. The workstations then use a stored latency number to decide what the time really is.

Client and server mode is for an environment that requires higher accuracy than broadcast mode. The client will periodically send messages to servers and will accept the information coming back from the server(s) to provide synchronization. Only the client can accept a synchronization and only the server can only give it.

## NTP Protocol

NTP is built on top of the User Datagram Protocol (UDP). The reason for choosing UDP over TCP is that a datagram protocol will not try and retransmit a lost packet. Since NTP packets are a perishable commodity, it is better to drop them than to delay and retransmit them. All NTP messages are basically the same, consisting of a header and some timestamps (see Fig. 8-6).

In the NTP protocol, actions occur in response to events. A typical event is when a timer for a particular peer expires. Other examples of events occur when a message is received from a peer, an operator command is entered, or a system fault occurs (e.g., the clock fails).

There are basically three procedures that are activated when an event occurs:

- Transmit
- Receive
- Update

A transmit procedure is called when a timer expires for a peer. Timers are kept for all peers in every mode except the server mode (the client initiates the transaction for client-server mode). When the timer expires, the NTP message is sent and the timer is reset. The number used to reset the timer is based on the network conditions in the path to that particular peer.

A receive procedure is called when an NTP message arrives. If the message does not match an existing peer-to-peer association, a new one is created. After doing a few sanity checks, the incoming message is used to calculate the round-trip delay of the message and the clock offset of the peer relative to this host.

In a peer-to-peer environment, messages are being continually exchanged back and forth. Each time a message is sent or received, it is timestamped. By looking at the messages in succession, one can calculate the round-trip delay.

The offset is also based on the timestamps that are continually exchanged. For a given pair of packets (one going from the host to its peer and then a response coming in), the clock offset can be approximated as follows:

- Take the amount of time that elapsed between the time this host sent a message and the time it was received by the peer. Note that we are looking at two different clocks here: this is one measure of the clock offset.
- Take the time elapsed between the time the peer sent a message and the time the host received it. This is another measure of clock offset.
- Take those two numbers and average them together.

Field	Length	Description
Leap Indicator	2 bits	Warns of an impending leap second to be inserted or deleted at the end of the current day. Values are:
		00: no warning
		01: last minute has 61 seconds
		10: last minute has 60 seconds
		11: alarm condition: local clock not synchronized
Version Number	3 bits	Currently 3
Mode	8 bits	Current operating mode
		0: unspecified
		1: symmetric active
		2: symmetric passive
		3: client
		4: server
		5: broadcast
		6: reserved for NTP control messages
		7: reserved for private use
Stratum	8 bits	1 is a primary reference, 2–255 are secondary servers. Stratum 0 is unspecified.
Poll Interval	8 bits	The interval between messages sent by a host to a peer. A host will always use the minimum of its own poll interval and the peer's poll interval. This field is indicated as seconds to the power of 2 (six is a minimum interval of 64 seconds)
Precision	8 bits	Precision of the local clock in seconds to the nearest power of 2.
Synchronizing Distance	32 bits	Estimated round-trip delay from the primary reference source in seconds with the fraction point between bits 15 and 16.
Synchronizing Dispersion	32 bits	Estimated sample dispersion to the primary reference source with the fraction point between bits 15 and 16.
Reference Clock Identifier	32 bits	For stratum 0 and 1.
	NIST	NIST public modem.
	ATOM	Atomic clock.
	callsign	Generic radio source.
Reference Timestamp	64 bits	The time the local clock was last updated.
Originate Timestamp	64 bits	The time when the last received NTP message was originated (copied from a transmit timestamp field upon arrival).

Field	Length	Description
Receive Timestamp	64 bits	Time when the last message was received.
Transmit Timestamp	64 bits	Time when the last message was transmitted.
Authenticator	96 bits	Optional. A key identifier and encrypted checksum of the message contents.

### 8-6 (Cont.) NTP Header

The clock offset numbers are checked for validity against the estimated maximum skew between the two clocks. Notice that both the round-trip delay and the offset are continually updated as more messages are exchanged.

The third procedure is the update procedure which is based on new delays and offset estimates. Before a local clock is updated, however, there is a weighted voting procedure based on the results from one or more peers. This is when the process of measuring samples and dispersion previously described occurs.

### *NTP Control Messages*

Most control over NTP system providers will be done using the Simple Network Management Protocol (SNMP) and extensions to the MIB database. Since NTP and SNMP services are implemented at different rates, there is an alternative to SNMP, NTP control messages. The control format includes provision for reassembly, since IP hosts are not required to reassemble packets larger than 576 octets and management messages may well be larger. Figures 8-7 and 8-8 show the format of the NTP control message. Figure 8-7 is the control message header, which indicates that there are a series of status words. Figure 8-8 shows the status words.

### Handling Warped Servers

A rule of thumb is that something is always broken. In the context of NTP, this means that a given server or client may be broken, and the protocol needs to be able to handle this problem. Several measures are built into the protocols to help deal with these problems:

- Reachability determination
- Authentication
- Sequence checking
- Adjusting polling rates



Field	Length	Description
Version Number	3 bits	Current = 3
Mode	3 bits	Value of 6 indicates an NTP control message.
Response bit	1 bit	0 for commands, 1 for responses.
Error bit		0 for normal response, 1 for error response.
More bit	1 bit	0 for last fragment.
Operation code	5 bits	0: reserved.
		1: read status command/response.
		2: read variables command/response.
		3: write variables command/response.
		4: read clock variables command/response.
		5: write clock variables command/response.
		6: set trap address/port command/response.
		7: trap response.
		8–31: reserved.
Sequence	16 bit	Used to match commands and responses.
Status	16 bits	Current status of the system, peer, or clock.
Association ID	16 bits	Integer identifying an association.
Offset	16 bits	Offset of the first octet in the data area.
Count	16 bits	Length of data field.
Data		Message data up to 468 octets.
Authenticator	96 bits	Optional.

### 8-7 NTP Control Message

Although NTP is built on the connectionless UDP, it makes sense to keep some state information on associations with peers. To determine reachability, a system simply keeps track of whether it is receiving an answer to a message it sends. Normally, every message gets an answer.

If no answer is received in between two transmissions, a single bit on an 8-bit register is set to zero and the bits are shifted to the left. Every time there are two successive transmissions, the procedure is repeated. If the register is all zeros, there have been eight consecutive transmissions and the peer is considered to be unreachable.

Authentication is an optional part of NTP. Having a remote user update local clocks could cause problems in the case of unauthorized, malicious users. A simple access control method is used in NTP that is based on matching the internet address with a specified range. This a simple access control measure, not authentication.

Field	Length	Description
System Status Word		Association Identifier is 0.
Leap Indicator	2 bits	00: no warning
		01: last minute has 61 seconds
		10: last minute has 50 seconds
		11: alarm condition
Clock Source	6 bits	0: unspecified
		1: calibrated atomic clock
		2: VLF or LF radio
		3: HF radio
		4: UHF satellite
		5: local net
		6: UDP/NTP
		7: UDP/TIME
		8: eyeball and wristwatch
		9: telephone modem
System Event Counter	4 bits	Number of system exception events since the last time a system status word was sent.
System Event Code	4 bits	The latest exception event.
		0: unspecified
		1: system restart
		2: system or hardware fault
		3: new status word (leap bits or synchronization change)
		4: new synchronization source or stratum
		5: clock reset
		6: invalid time or date
		7: clock exception condition
Peer Status Word		In response to a command.
Peer Status	6 bits	Status of the peer with each bit a flag.
		0: configured
		1: authentication enabled
		2: authentication okay
		3: reachability okay
Peer Selection	3 bits	Status of the peer based on time provided.

Field	Length	Description
		0: rejected
		1: passed sanity checks
		2: passed correctness checks
		3: passed truncation checks
		4: passed outlier checks
		5: current synchronization selection
		6: current synchronization source
Peer Event Counter	4 bits	Number of events since last peer status word returned.
Peer Event Code	4 bits	Last peer exception event.
		0: unspecified
		1: peer IP error
		3: peer unreachable
		4: peer reachable
		5: peer clock exception
Clock Status Word		Indicates the host's system clock or some peer clock as indicated by an association ID.
Clock Status	8 bits	0: clock operating within nominals
		1: reply timeout
		2: bad reply format
		3: hardware or software fault
		4: propagation failure
		5: bad date format or value
		6: bad time format or value
Clock Event Code	8 bits	Identifies the latest clock exception event.

**8-8 (Cont.) NTP Status Words**

In highly secure environments, the optional cryptographic authentication procedure can be used. Using DES encryption and a checksum ensures that a message is from the correct source and that it has not been altered.

Sequence checking is the third mechanism to help assure robustness. This mechanism is a series of sanity checks on the timestamps inside a message. If the transmit timestamp of a received message is the same as a previously received transmit timestamp, for example, we can assume the message is a duplicate. Additional checks can make sure that timestamps are monotonically increasing to prevent against out-of-order packets.

Careful control over polling rates is the fourth mechanism used to assure robustness. Primary time servers can have 300 or more peers. To prevent an excessive load, it is desirable to have a fairly long polling interval. At first, the polling intervals for a particular association are very quick to make the clocks synchronize. Polling intervals will be as short as a minute at the beginning of an association and will grow to 17 minutes as the clocks become closely synchronized.

### Implementation in the Internet

In the Internet, there are a series of computers that act as primary time servers in Canada, Germany, Norway, the United Kingdom, and the United States. The primary servers connect directly into national time services. In May of 1991, there were at least 58 such primary servers. Many of these computers were directly on national backbone networks and were available to all users. The remaining were on regional networks and were intended to be used within the regional network environment.

The primary servers continuously exchange messages with each other, providing a high level of redundancy. If a timecoder on a particular primary server fails, that server is able to use one of its peers as a source. The primary server thus becomes a secondary server, connecting to the primary server that has the lowest available stratum (1 in this case) and the smallest synchronizing distance.

The secondary time servers on the Internet consist of several thousand computers. More robust configurations have a set of servers agree to provide backup to each other. The University of Illinois and the University of Delaware provide such an active backup system with each other.

### *Does it Work?*

To test the accuracy of time servers in the Internet, David Mills used each of the three time-request protocols (ICMP, TIME, and NTP) to every Internet address he could find. Using a list compiled at SRI International, a total of 112,370 hosts were found. His results, summarized here, are detailed in an article in RFC 1128, later updated in an article in the *Computer Communications Review* (see "For Further Reading").

For each entry in the file, the experiment tried using each of the three protocols to each host. If no reply was received in 1 second, the program tried again. If no reply was received after another second, the attempt was abandoned. If an ICMP host or network unreachable message was received, the attempt was also abandoned.

The program ran on a host that was synchronized within a few milliseconds of the NIST clock. For each of the three time protocols, offsets for up



to eight replies were accumulated for each host. The maximum, minimum, and mean were then put into a data file.

Out of 107,799 hosts that were ultimately surveyed, 94,260 resulted in some kind of an entry (i.e., an error message or data). Of this total number of hosts, 22 percent (20,758) returned timestamps that might be valid.

The protocols were ranked so only one was used for a particular host. NTP was the first preference, TIME next, and ICMP third. After further pruning, the list was reduced to 8455 hosts: 3694 provided ICMP timestamps, 7666 responded to the TIME protocol, and 789 responded to NTP requests. In addition to the 789 hosts, the NTP synchronization subnet was searched using a monitoring program which added a few more NTP hosts for a total of 946.

### Results

The results were fairly dramatic. In the case of NTP, the hosts were fairly accurate. Only eight hosts had errors above 10 seconds and it was assumed that these hosts were using old versions of NTP. At least 30 percent of the NTP hosts kept time within 30 milliseconds.

ICMP and TIME are significantly less accurate. One percent of the ICMP hosts had errors greater than a day and 1 percent of the TIME hosts had errors greater than a few hours. Thirty percent of ICMP hosts kept errors to a couple of minutes and 30 percent of TIME hosts kept errors to within 1 minute. As a rule, NTP maintains a clock accuracy 3 orders of magnitude (1000 times) over the other protocols.

A further study was made within the NTP synchronization subnet. Over a period of 18 months, surveys were made of six primary time servers located throughout the country. The paths between these servers could be quite long—up to 12 network hops in some cases. The median error over a large number of samples was only a few milliseconds, while the maximum error was no more than 50 milliseconds.

### For Further Information

D. L. Mills, Internet Time Synchronization: the Network Time Protocol. Forthcoming in *The IEEE Transactions on Communications*. Available by FTP on pub/ntp/trans.ps.Z on louie.udel.edu.

———, Measured Performance of the Network Time Protocol in the Internet System, RFC 1128, October, 1989.

———, Measured Performance of the Network Time Protocol in the Internet System, *Computer Communication Review* 20, 1 (January 1990), pp. 65-75.

———, Network Time Protocol (Version 3), Internet Draft Standard, December, 1990. Available by FTP on pub/ntp/ntp3.ps.Z on louie.udel.edu.



# Names





# Names

## The New-Name Naming Service: ~~Yellow Pages~~ NIS

The original naming service in the Sun ONC protocol family was called the Yellow Pages. Through some oversight, Sun marketing forgot to check and see if they could in fact use the name—after all you can't use other people's names, especially in a naming service. Otherwise, it would be very simple to call a new computer line "MicroVAX VI." In the case of the Yellow Pages, it appears that British Telecom objected in no uncertain terms. So, recent manuals all refer to the Sun naming service as the Network Information Service (NIS). Of course, the commands still tend to begin with the prefix "yp."

The original impetus for NIS was the problem of proliferating system files. Take the `/etc/passwd` file, for example. In a distributed environment with lots of NFS servers and lots of workstations, it makes sense to have a unique user namespace across the network. Since each computer has its own `/etc/passwd` file, in order to have a unique namespace there needs to be some way to keep all the files synchronized. Of course, the original method for doing this was through complicated shell scripts written by system managers which copied the "official" copy to all other hosts on the net—each site having its own system and most systems working only through a great deal of effort.

What NIS does is provide a mechanism to distribute the name keeping service. The basic service takes a key and returns a value. A key might be a username and the value an encrypted password. Another key might be a host name, and the corresponding value the IP host address.

Each translation that NIS provides is known as a map. In the case of some maps, there are actually two services. A host might want to translate a host name into a host address—given a mail message addressed to "user@host," the client needs to find an internet address corresponding to the SMTP port on that host. Or, a client might wish to translate a host

address into a host name. Given an incoming SMTP connection on an IP source address, the client may wish to know the host name in order to fill out a “Received From” header line in the mail message.

NIS takes a map and makes it available to all nodes on the network. NIS coordinates updates to the map and provides a mechanism whereby multiple NIS servers can all look up answers in the same map—this replication thereby increases both the availability and the speed with which a client can have a lookup returned.

This chapter will start with a description of NIS. Next, the successor to NIS, NIS+ is examined. NIS+ provides a much more sophisticated service than NIS, while still preserving compatibility with the NIS maps. Next, the Domain Naming System (DNS) is described.

NIS is meant to work among a group of Sun or ONC hosts. NIS is optimized in many ways for the workgroup or enterprise network, although it can be used for very large networks. DNS, by contrast, is meant to work in the decentralized, wide-area environment of the Internet. DNS doesn’t provide as many services as NIS+ or NIS, but it does enable multiple administrative domains to all provide services to each other.

## NIS

NIS is a simple distributed name service for administering basic system administration files. It also allows, through the use of the public key map, the running of Secure RPC and Secure NFS across the network.

NIS provides a fairly loose guarantee that, within a domain of computers, names will be unique. It allows a client to look up a value associated with that name. The concept of a loose uniqueness guarantee is an important one. NIS is not a distributed database server, it does not provide a generalized lookup service, and it does not guarantee instantaneous convergence of multiple values.

A domain is simply a set of computers. A network may have several domains. The Sun Wide-Area Network, for example, consists of several different domains. While the entire network could have been put into one domain, segmentation simplifies administration of the namespace. NIS does allow lookups in alternative domains, but operations default to the current domain.

A map is the data that is kept by the NIS servers. Most maps originate as a text file and are transformed into NIS format using a special make file. Each map has one master server that is responsible for all updates.

Maps may also have secondary servers. A secondary server has a copy of the map, which may or may not be up to date. Over time, NIS guarantees that updates to the master copy of the map are propagated or “pushed” to the secondary servers. This loose consistency makes NIS ideal for an envi-

ronment in which data does not change rapidly. Rapidly is a relative term, of course, but in this case we refer to files that change over minutes and hours instead of milliseconds and seconds.

All ONC clients have the ability to consult these maps by sending NIS requests to a server. To find a server, a special process called ypbind is used. Ypbind is responsible for finding a server. Once a bind is made, a variety of system calls and user utilities can be used to consult maps.

### *Components of NIS*

There are five basic elements to NIS:

- Domains
- Maps
- Daemons
- Utilities
- Maintenance commands

A domain is just a set of maps that are in common. Each domain has a domain name and every computer is in only one domain.

The daemons are the processes that run on clients and servers, while the utilities are the programs that run on the client systems. Maintenance commands are used by network administrators to make maps and force the propagation of updates (see Fig. 9-1). Note that in addition to the user commands, programmers can access NIS functions through the use of library routines.

### *Maps*

Maps are kept on servers in the directory:

```
/var/yp/domainname/ ....
```

If you look in /var/yp of a master server, you will see a makefile, which in turn calls makedbm to create the maps.

Makedbm is the program that actually makes the map. It takes as input a key/value pair where key is one word and value is anything that follows on that line. The result is two files:

```
/var/yp/domainname/mapname.dir  
/var/yp/domainname/mapname.pag
```

There are a wide variety of different maps used in a Sun network. Figure 9-2 shows some of the default maps. In addition, the programmer or network administrator may develop new maps for other applications.

Notice that many of the maps replace or supplement the files that are in the /etc directory. A typical example are the hosts and password files. In



Daemons	
ypserv	Server process.
ypbind	Binding process.
ypxfrd	Map transfer daemon.
rpc.yupdated	Server for changing map entries.
rpc.yppasswdd	Server for modifying the NIS password file.
in.named	Optional daemon used in a DNS environment.
Utilities	
ypcat	List data in a map.
ypwhich	List name of my NIS server and the maps served.
ypmatch	Match a key to its value in a map.
ypinit	Build and install a database.
yppoll	Get a map's order number from a server.
Auxiliary Utilities	
yppush	Propagate from master to slave servers.
ypset	Set binding to a particular server.
ypxfr	Transfer data from master to slave server.
makedbm	Create dbm file for an NIS map.

## 9-1 NIS Components

order for some utilities to take advantage of the NIS services, it was necessary to rewrite some common Unix utilities. The `passwd` command, for example, allows users to change their passwords. The command does so by editing the `/etc/passwd` file. The `passwd` command is written as a `setuid` command—it gives the user superuser privileges but only for the purpose of changing one line in the file.

When the `/etc/passwd` file is replaced by a password map, it is necessary to rewrite the `passwd` command so that it knows to look for passwords in both the file and the NIS map. Likewise, the login process needs to know to issue NIS calls to maps as well as to check the local file.

Most maps are maintained in text files on the master server. Updates to the maps consist of editing this text input file and then running the `make` utility. To find out which server is the master for a particular map, the user runs the `ypwhich` command:

```
mynode% ypwhich -m map_name
```

Here `map_name` is the name or nickname for the map. The result is the name of a host on the network.

In many situations, nine different `/etc` files can be replaced in whole or in part by NIS maps:



Map Name	Source File	Description
bootparams	boot params	Pathnames of files clients need while booting such as root and swap: replaces /etc/bootparams.
ethers.byaddr	ethers	Machine names and addresses: replaces /etc/ethers.
ethers.byname	ethers	Same as above but index is by machine name.
group.bygid	group	Group security information with group ID as key.
group.byname	group	Name is the index.
hosts.byaddr	hosts	Host name and IP address with IP address as key. /etc/hosts is consulted at boot time.
hosts.byname	hosts	Same as above with host name as key.
mail.aliases	aliases	Aliases and mail addresses with aliases as the key. The local /etc/aliases is first consulted.
mail.byaddr	aliases	Same as above with mail address as the key.
netgroup.byhost	netgroup	Contains group name, username, and host name (with host name as key). Replaces /etc/netgroup.
netgroup.byuser	netgroup	Same as above with username as key.
netgroup	netgroup	Group is the key.
netid.byname	passwd, hosts, group	Used for Unix-style authentication. Contains a machine name and mail address (including the domain name). The netid file is consulted first.
netmasks.byaddr	netmasks	Network mask used with IP subnetting. /etc/netmasks not consulted.
networks.byaddr	networks	Names and IP addresses. Replaces /etc/networks.
networks.byname	networks	Name of network is the key.
passwd.byname	password	Username is the key. Supplements /etc/passwd.
passwd.byuid	password	User ID is key.
protocols.byname	protocols	Network protocols known. Replaces /etc/protocols.
protocols.bynumber	protocols	By protocol number.
publickey.byname	publickey	Used for secure RPC. Replaces /etc/publickey.
rpc.bynumber	rpc	Program numbers and names known to RPCs. /etc/rpc is replaced.
services.byname	services	Internet services. Replaces /etc/services.
ypservers	none	Known NIS servers.

## 9-2 NIS Maps

- Passwd
- Hosts
- Ethers
- Group
- Networks
- Protocols
- Services
- Aliases
- Netmasks

The `passwd` file is an example of a file that should be supplemented by the NIS map, and not necessarily replaced entirely. Programs first go to the local `/etc/passwd` file to check for a user and then check the NIS maps. Every client should have a local password file with local entries for root and primary users. Local is a relative term as the file may actually reside on an NFS server if this is a diskless node. At the end of the file, an escape sequence is used to force a search of the NIS password database:

```
root:9wxntql2tHT.k:0:1:Operator:/:/bin/csch
myname:7kjDXZD/Hug2s:624:20:myname:/home//myname:/bin/csh
+;
```

The `hosts.equiv` file is another file that might be replaced by an NIS map. Normally, `hosts.equiv` has a single line with only the plus character in it. This escape sequence means that anyone in the password database can access the machine from any known host: this is the default.

Better security can be provided with `netgroups`. Putting `+#` or `-#` in the `hosts.equiv` file tells a machine to use the NIS `netgroups` database to decide how to handle permissions.

The `/etc/hosts` file is one that needs to be individually maintained on each system. Even if the machine is using the NIS service, there needs to be a local host name. On a diskless machine, the file should also contain the name of the boot server. Once `ypbind` is up and running (after booting is complete) `/etc/hosts` is no longer used and the NIS hosts database takes over.

Other files, such as `groups`, are easily maintained strictly as NIS maps. In order to inform the user programs to consult the maps, on a client the entire file is reduced to:

```
+;
```

In addition to the default maps, most sites maintain a wide variety of other information in NIS. Figure 9-3 shows typical output from the `ypwhich` command, which is used to find out the current status of NIS.

The first command, `ypwhich` with no parameters, shows that the server named `Commserve` is now the default server for the workstation `Hydramatic`. The next command shows which servers are acting as the master server for particular maps. Notice the wide variety of different maps kept (and the figure shows only a small portion of the actual maps used). Finally, the last `ypwhich` command shows nicknames used to refer to certain maps.

### *NIS Servers*

Servers are the systems that do the bulk of the NIS work. There are two types of servers: masters and slaves. The master server is the one that han-

```
hydramatic% ypwhich
commserv
```

```
hydramatic% ypwhich -m
hosts.byaddr parkway
sg.auto.home parkway
sg.auto.db parkway
sg.auto.design parkway
mail.aliases parkway
APPCs.byname comic
sparcdudes.master parkway
auto.master parkway
auto.nse moon
timezone.byname parkway
netgroup.byuser parkway
appcs.byname parkway
resources parkway
networks.byname parkway
products parkway
netgroup parkway
netgroup.byhost parkway
rjes.byname parkway
auto.dump parkway
protocols.byname parkway
ypservers parkway
3270s.byname parkway
ethers.byaddr parkway
passwd.byuid moon
nse.branch.byid moon
ethers.byname parkway
nse.global.byname moon
passwd.byname moon
netid.byname moon
rpc.bynumber parkway
services.byname parkway
bootparams parkway
mail.byaddr parkway
hosts.byname parkway
group.bygid parkway
publickey.byname moon
```

```
hydramatic% ypwhich -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
```

dles updates and propagates them to slave servers. Slave servers have a read-only copy of maps.

As a general rule, an NIS server is often the same machine as an NFS file server. This is not required, of course, but it makes sense to combine the two functions. Likewise, as a general rule, the gateway to the backbone is probably not a good candidate to double as an NIS server, at least as the master server. Times when the gateway is under heavy load may often coincide with times that the name server is under stress. Peak problems are thus compounded by putting the two functions on the same machine.

The slave server has a complete copy of the master's maps. Slaves are needed for two reasons:

- Slaves improve performance by distributing the query load over multiple systems.
- Slaves improve availability by keeping duplicates of crucial system files.

Since NIS uses broadcasts, it is important that there is at least one server, slave or master, on each subnetwork. This is because broadcasts are not forwarded across gateways in the TCP/IP environment. Typically, two or three servers are available on each local segment to provide availability. If one NIS server becomes inaccessible, a client will automatically rebind to another available server.

### *Binding*

Binding is the process by which a server is picked. Ypbind is the command that handles this. Once the binding is established, the client makes a call to the ypserv daemon on the server. Note that the binding between server and client can change in accordance with network loads. The ypwhich command tells which server is handling things at the time:

```
my_node% ypwhich  
yp_server
```

This binding process is the reason for requiring a server on each subnetwork. Ypbind issues an RPC broadcast, which in turn contains the IP broadcast address to reach all hosts on the subnetwork. The first node to respond to the client request becomes the client's server. If for some reason that server becomes unavailable, the broadcast is repeated and a new server picked.

Even though binding is limited to a subnetwork, that doesn't mean that an NIS domain is limited to a subnetwork. The process of moving maps back and forth does not use the broadcast mechanism, so servers can communicate across gateway boundaries.



## *Starting up Systems*

Starting up a master server is simply a matter of running the initialization script and then starting up the server. As a general rule, these commands are put into `rc.local` to be run every time the master server boots:

```
# cd /var/yp
# /usr/etc/yp/ypinit -m
# ypserv
```

On a master server, the `ypxfrd` high-speed transfer daemon would also be started up.

A slave server needs to run the same steps but needs to precede that with finding the name of the master server:

```
# cd /var/yp
# ypbind
# /usr/etc/yp/ypinit -s master_name
# ypserv
```

The client system is quite simple: all the user needs to have is the `ypbind` command someplace so that the server can find the name of the relevant server.

As a general rule, NIS is transparent to the user. An exception is the `passwd` command. When users change their passwords, they are changing a map. A special `yppasswd` command creates new passwords in the maps. For this to work, the `rpc.yppasswdd` daemon should be running on the master server.

Updating the map on the master and having that update show up on a slave's copy of the map are two different things. As a general rule, users should make their changes before going home; updates will have fully propagated by the next day.

Propagation of maps is something that is only done periodically. When a slave does a `ypinit`, this forces a full transfer of maps. After that, the `ypxfr` command needs to be run. This command is usually put into the `crontab` files on servers, which will then run periodically, usually every night.

The `crontab` on each slave server contains a series of `ypxfr` entries. Each time `ypxfr` activates, it contacts the master server and only does a transfer if the master has a more recent copy of the map:

```
/usr/etc/yp/ypxfr -h master mapname
```

The `ypxfr` command works in tandem with another, called `yppush`. `Yp`-push checks the `ypservers` map, contacts each slave server in the list, and sends it a "transfer map" request. When the request is ACKed by the slave, the slave does the actual transfer by using the `ypxfr` program.

Leaving the request for the new map up to the client means that if a slave is down, the system will continue to work. If a master is down, the slaves will be unable to get new changes but can continue serving old data.

The `ypxfr` command has a special `-s` option, which allows a map to be received from another domain. This is how Sun has handled a “master master” database. The master for a domain goes to the “master master” to get centrally administered files. In this way, each domain gets a separate master server, but the master servers can work together.

## Typical NIS Operations

Figures 9-4 through 9-6 show typical operation of the NIS service. In Figure 9-4, the first operation is a lookup operation in the domain Sun, using the address as the key. Figure 9-5 shows the result of that operation, the name Wimpy.

We also see in Figure 9-5 the continuation of this operation, a lookup in the `hosts.byname` map in order to find the internet address of the host Wimpy. Once this address is found, we see in Figure 9-5 that the client node is setting up a TCP connection to Wimpy (notice the synchronization bits set on the next two TCP packets). Packet number 9 shows us the reason for all this traffic: an NFS operation to get the attributes of a file.

The last diagram, Figure 9-6, shows an example of a map operation to do a repeated lookup, in this case a group lookup in a domain called Dayton. The powerful `get next` operation allows repeated queries to a map. We will see that NIS+, the successor to NIS, performs a similar row-by-row retrieval, but in a much more flexible manner.

## NIS+

NIS is quite effective for the limited role set out for it: maintenance of common systems files for use in fairly small networks. While NIS has managed to scale up to very large networks (most notably the 10,000 nodes on Sun’s internal network), it lacks features of some more modern services.

The successor to NIS is NIS+. NIS+ is a lightweight name service, which means that it is not intended as a full-fledged database. It is thus a complementary service to X.500: X.500 is a full-fledged attribute-based naming system. NIS+ is intended to provide a more limited subset of functionality, but at a much higher speed than X.500. In order to understand some of the NIS+ changes, we can start with four limitations of NIS:

- Flat namespace
- Objects are character strings only
- Minimal access control and authentication
- No updating capabilities

SUMMARY	Delta T	DST	SRC						
M 1		Sun	01C0C2+Sun	01B595	YP	C	Lookup key sun.nsup.byaddr		
2	0.1353	Sun	01B595+Sun	01C0C2	YP	R	OK		
3	0.0079	Sun	01C0C2+Sun	01B595	YP	C	Lookup key sun.hosts.bynam		
4	0.1243	Sun	01B595+Sun	01C0C2	YP	R	OK		
5	0.2068	Sun	01C0C2+Sun	01B595	TCP	D=2001 S=1100	SYN SEQ=379705		
6	0.0027	Sun	01B595+Sun	01C0C2	TCP	D=1100 S=2001	SYN ACK=379705		
7	0.0019	Sun	01C0C2+Sun	01B595	TCP	D=2001 S=1100	ACK=374137		
8	0.0034	Sun	01C0C2+Sun	01B595	TCP	D=2001 S=1100	ACK=374137		
9	0.0202	Sun	01B595+Sun	01C0C2	NFS	C F=5D63	Get attributes		

## DETAIL

```

YP: ----- SUN Yellow Pages -----
YP:
YP: Proc = 3 (Return value of a key)
YP: Domain = sun
YP: Map = nsup.byaddr
YP: Key = 200
YP:
YP: [Normal end of "SUN Yellow Pages".]
YP:

```

Frame 1 of 15

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 9-4 An NIS Query

SUMMARY	Delta T	DST	SRC						
M 1		Sun	01C0C2+Sun	01B595	YP	C	Lookup key sun.nsup.byaddr		
2	0.1353	Sun	01B595+Sun	01C0C2	YP	R	OK		
3	0.0079	Sun	01C0C2+Sun	01B595	YP	C	Lookup key sun.hosts.bynam		
4	0.1243	Sun	01B595+Sun	01C0C2	YP	R	OK		
5	0.2068	Sun	01C0C2+Sun	01B595	TCP	D=2001 S=1100	SYN SEQ=379705		
6	0.0027	Sun	01B595+Sun	01C0C2	TCP	D=1100 S=2001	SYN ACK=379705		
7	0.0019	Sun	01C0C2+Sun	01B595	TCP	D=2001 S=1100	ACK=374137		
8	0.0034	Sun	01C0C2+Sun	01B595	TCP	D=2001 S=1100	ACK=374137		
9	0.0202	Sun	01B595+Sun	01C0C2	NFS	C F=5D63	Get attributes		

## DETAIL

```

YP: ----- SUN Yellow Pages -----
YP:
YP: Proc = 3 (Return value of a key)
YP: Stat = 1 (OK)
YP: Value = Winpy
YP:
YP: [Normal end of "SUN Yellow Pages".]
YP:

```

Frame 2 of 15

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 9-5 Query Results



The screenshot shows a network analysis tool interface. At the top, there is a 'SUMMARY' section with a table of network events. Below this is a 'DETAIL' section showing information for a specific event (Frame 118). At the bottom, there is a toolbar with buttons for various functions.

SUMMARY	Delta T	DST	SRC	YP	C	Get first entry dayton.mea
117	0.2683	Sun	06E6F5+Sun	076CF9	YP	R OK
118	0.0062	Sun	076CF9+Sun	06E6F5	YP	C Get next entry dayton.mea
119	0.0119	Sun	06E6F5+Sun	076CF9	YP	R OK
120	0.0049	Sun	076CF9+Sun	06E6F5	YP	C Get next entry dayton.mea
121	0.0103	Sun	06E6F5+Sun	076CF9	YP	R OK
122	0.0048	Sun	076CF9+Sun	06E6F5	YP	C Get next entry dayton.mea
123	0.0100	Sun	06E6F5+Sun	076CF9	YP	R OK
124	0.0048	Sun	076CF9+Sun	06E6F5	YP	C Get next entry dayton.mea
125	0.0102	Sun	06E6F5+Sun	076CF9	YP	R OK

**DETAIL**

YP: ----- SUN Yellow Pages -----

YP: Proc = 4 (Get first keyu-value pair in map)

YP: Stat = 1 (OK)

YP: Key = dts-sd-team:\*:102:daveb,jmc,curt,slee,dem,gordon

YP: Value = dts-sd-team

YP: [Normal end of "SUN Yellow Pages".]

YP:

Frame 118 of 339

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 9-6 The Powerful Get Next Operation

The flat namespace means that a map is available to the entire domain. It also means that servers cannot know information about other domains. There is no provision for subdomains which contain a subset of the namespace. NIS+ addresses this problem by adding hierarchy to the namespace.

In NIS, the value of an object is a simple character string. While adequate for the original NIS goals, there are several instances where a client will want to store arbitrary data in a map.

NIS has no access control. The basic NIS service is read-only access to data, with no security provisions: everybody can read everything. There is no concept of private maps or access control lists for maps. In the case of the password map, for example, a user can look at the (encrypted) passwords of all users. This is not necessarily a large security hole, but it certainly makes the system more vulnerable to crackers. (Note that a solution to this particular problem is provided in the C-2 security enhancement to the Sun operating system.)

Finally, NIS has a problem of difficult administration. The service is only a lookup service, and there are no provisions for modification of data. This means that modifications must be made out-of-band—logging into a master server, editing a text file, and then running the make utility.



We start first with a brief overview of NIS+ and will then proceed to discuss each of the points in more detail. NIS+ provides two basic services:

- A distributed name service
- A database management system

As a namespace provider, NIS+ lets the user add, remove, and look up names. A lookup operation resolves a name to a specific instance of an object.

Databases that NIS+ manipulates are named within the hierarchical namespace. As a database provider, NIS+ provides a series of primitives for the manipulation of records in the tables: add, remove, search. Note that tables are named objects in the namespace, but records are not directly named.

The NIS+ service is provided by a set of name servers. Each server handles a portion of the namespace. With the exception of the root server, a server handles the portion of the namespace underneath it.

The database contents of each name server may be replicated in one or more service points. One of the service points is the primary service point: it is the contact point for principals that want to make changes to the namespace.

NIS+ clients are called principals. Each principal has a public key and a private key pair, which are used for authentication purposes. Accessing information about a principal from NIS+ requires the name of that principal and its public key. This entry point to NIS+ may be further limited by access rights defined for that principal.

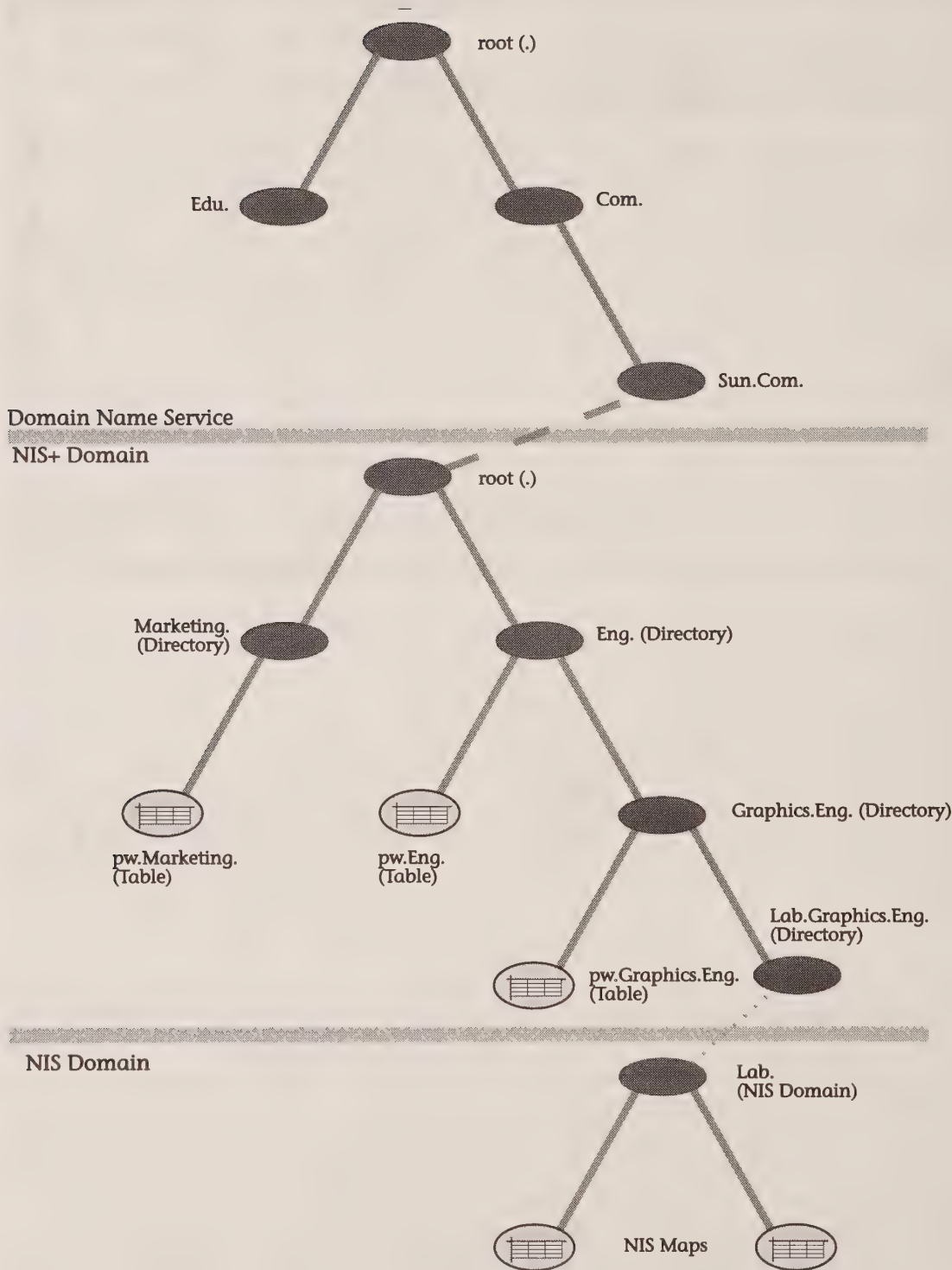
Just as the target of a query may be a principal, so may the source. The source, a username on a machine, is identified by a name and the same key pair. A host could use this information to retrieve information about a user such as the encrypted password associated with that username for login purposes.

## *Names*

NIS+ names consist of a series of labels, each corresponding to a portion of the directory tree (see Fig 9-7). The top part of the hierarchy is called the root. The root is a type of object called a directory. The directory has as its contents a series of names, each being an object.

Below the root are further directories. Eventually, at the bottom (or leaf) of the tree, there are other kinds of objects. This is analogous to a file system: it is a series of directories, terminated by a name.

In NIS+, there are a variety of different kinds of objects at the leaf of the tree. One kind of object is the table, which is equivalent to the NIS map but is greatly enhanced. There are other kinds of objects: a symbolic link, for



## 9-7 NIS+ and Other Name Services

example, is a pointer to some other part of the name tree and is thus a method for providing aliases.

In NIS, all objects are in a common domain. The default for any search is the current domain, although it is possible to search another domain if you know its name. This is an example of a flat namespace: a root and a map.

In NIS+, we can organize the namespace. The root might be Engineering. Under that root are subdomains:

```
Graphics.Engineering
FDDI.Engineering
Database.Engineering
```

Since Graphics is a directory, it has a series of objects in it. These objects might be the traditional NIS maps: passwd, hostname, groups, and other system administration files. An example would be:

```
Passwd.Graphics.Engineering
```

The number of subdomains can be very large. Instead of Graphics being the final subdirectory, there might be lower-level names corresponding to individuals, each with its own private databases.

Since objects can be at any level of the hierarchy, each subdirectory might have a password file:

```
Passwd.Malamud.Graphics.Engineering
Passwd.Graphics.Engineering
Passwd.Engineering
```

One more twist on the namespace: the root might actually be a pointer to some other naming service. Engineering has been used as the root domain. One could presume that Marketing would insist on also being a domain. Both of those can be part of the organization-wide name system:

```
Passwd.Malamud.Graphics.Engineering.Sun
```

The objects under Sun—Engineering and Marketing—are managed by DNS, another naming system. In fact, we will see that Sun is, in turn, shorthand for Sun.Com, adding another level to the naming hierarchy. A central name server for the Internet handles the top level directory: Com. Another DNS server handles the subdomains of Sun. When we get down to the level Engineering, the namespace is handled by an NIS+ name server.

This concept of a long hierarchy may seem to make for long names. In reality, most systems use the concept of a default. In the case of local look-ups, for example,

```
Malamud.Graphics.Engineering.Sun.Com
```

would be the default. A lookup to “passwd” would be the local password file. Asking for “passwd.FDDI” would request the file in the FDDI subdirectory.

The power of this hierarchy is that it allows access to names in a very broad environment. If the local name server can’t handle the issue, requests are made to servers higher in the tree.

Where to make the split between the NIS+ domain and some other system is up to an organization. In our example, Engineering would be the root for what NIS+ would handle. An example query target would be:

Passwd.Hyperbole.Marketing.Sun

The local NIS+ server would send the query up to the Engineering server, which would see that the Sun directory is outside its scope. The Engineering server would send the client program back a message indicating that the Sun domain is handled by DNS. DNS would provide the location of the Marketing.Sun server. The client would then switch back to NIS+ and attempt to do the lookup on the Hyperbole subdomain.

Switching between NIS+ and DNS (and X.500 and other systems) requires a smart set of clients. Another alternative is to use NIS+ to do as much as possible. Here, the line between DNS and NIS+ would be drawn on top of Sun: any requests for DEC.COM would be referred to DNS. Any issues within the Sun.Com namespace would be handled by NIS+.

A name is thus a series of labels separated by dots. A label is a string of characters that does not normally include special characters such as the period, double-quote, colon, percent, or square bracket. Leaving out the colon and the percent sign from normal NIS+ names mean they can coexist with names in Digital’s DNA Naming Service. More on this later.

If it is necessary to use special characters in a name, double quotes are used:

“Any.Bad%Awful:Name”.Hyperbole.Marketing

The quotes hide the inside of the label from the name service. Names like this are particularly bad because of the number of special cases and ambiguities that arise from mixing the syntax from different naming services.

### *Objects*

Eventually, we will reach the leaves of a namespace hierarchy and see the basic object: the database object of NIS+. The NIS map is a special case of a database consisting of a key with a single value.

In NIS+, the map has become a table. The table is a structure with a fixed number of columns and a variable number of rows. Some columns can be searched and have names. Others just hold data.



The NIS map is really a two-column table, in which one of the columns can be searched. In the NIS world, to make it possible to search both columns, two maps are generated: `hostnames.byaddr` is an example where one map is generated for searching by address, another for searching by name. These dual NIS maps can be replaced with a single NIS+ table in which both columns can be searched.

To refer to individual rows of a table, the concept of an indexed name is used. The indexed name is the name of the column and a search value:

```
( username = Malamud ), Passwd.FDDI.Graphics
```

The search criteria can yield one or more rows. This particular search, of course, will probably yield only one username (and its associated encrypted password).

Doing a lookup in a database is the equivalent of the basic NIS operation. In addition, NIS+ enables users to make updates to the database by creating, deleting, and modifying objects.

Objects are directories, tables, and pointers. Each of these is represented in NIS+ as a data structure that has a common set of properties and a variable part that is unique to each type of object.

Common properties are things we want to know about all objects. Examples of these common properties are things like which owner or group the object belongs to, object creation time, time of last modification, access rights, and object type.

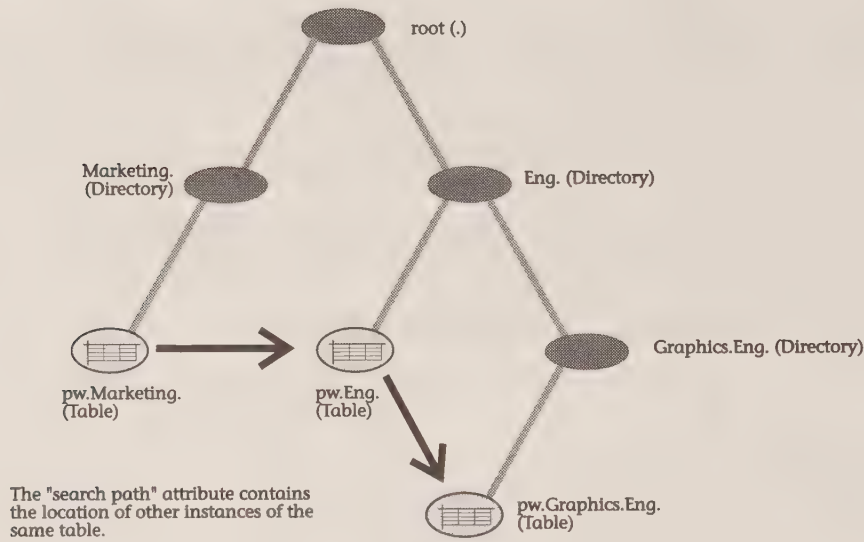
The variable part depends on the object type. For a table, the variable part would include the names for columns, if they can be searched, and the object type and structure of the entries.

Inside the table are a series of cells, representing the intersection of a row and a column. Since each column can be an NIS+ object, it is possible to put directories inside tables or even tables inside tables. Within a given column, all objects are of the same type.

Tables are one basic type of leaf object, but neither the rows nor the columns of a table have NIS+ names, only the table itself does. Rows can be accessed via indexed names. An entry (e.g., a cell) is really an object: it has a data structure that is like an object. It has information in the fixed part such as ownership, type, and structural information.

Not all columns of a table have names. Some columns cannot be searched, and for these columns the name is optional. This is an example of how NIS+ differs from a general-purpose database, where all columns of a table are always named. This reduction in generality helps make NIS+ faster than a full-fledged database system since it doesn't have the overhead of supporting truly arbitrary searches.

As a general rule, the entries in a table consist of either a piece of data or a link object. The link object is simply an NIS+ name: a pointer to another



9-8 NIS+ Tables and the Search Path

portion of the namespace. It is also possible to link a series of tables together using a search path (see Fig. 9-8). After one table is searched, another table is pointed to. This is quite useful when there is a need for fragmentation of a database, as in the case of a passwords database. While users want to see a common, network-wide view of passwords, usernames like root need to be locally administered for each machine, requiring a fragmented database.

There is one more type of object: groups. A group is a set of NIS+ principals (computers that use NIS+ and have names) and their associated groups. Groups can be recursively resolved to a set of principals (and, it is hoped, not recursively resolved back to the same group).

### NIS+ Services

In NIS, there is one service: look up the single data value associated with the key. NIS+ has a much larger number of operations:

- Namespace operations
- Table operations
- Group operations
- Replica operations
- Navigation
- Object verification

Namespace operations operate on the namespace itself. The most common operation is the lookup operation: the user gives NIS+ a name and

NIS+ returns a copy of the object. In the case of a directory, we get back a set of NIS+ names. As we look up each of those names, we get back the objects: more directories, tables, links, and groups.

The table operations are typified by the list operation, a command which allows the user to search the table specifying one or more attributes as keys. Other commands can add, remove, and modify the table.

An NIS+ client is able to accept multiple rows to be returned for a list operation. In NIS, only a single row would be returned at a time. To keep compatibility, `first_ibase` and `next_ibase` operations are provided.

The group operations includes the enumerate operation, which returns all the members of a group. Another useful operation is `is_member`, which verifies if a principal is a member of a group.

Replica operations allow servers to move information back and forth. A receive operation transfers update information between primary and secondary service points (masters and slaves in the NIS terminology). A ping operation notifies another system of an update. A dump operation duplicates an entire directory.

One big difference between NIS and NIS+ is the ability in NIS+ to move updates incrementally instead of transferring whole maps. This makes update propagation significantly more rapid than in NIS, particularly when maps get very large.

Navigation operations are helpful for large namespaces. The `find_directory` operation takes a name and returns the location of a name server. This is usually the first operation that is issued for any objects that reside in nondefault domains.

The last operation is used to provide consistency when a node has a cache of an object. The `is_newer` operation compares two copies of an object and determines which is the current version.

There are really four kinds of lookup commands here. The `find_directory` command finds a server. The `lookup` command is used to return an object once a server has been found. The `list` command is used inside tables, as the `enumerate` operation works inside groups.

## Security in NIS+

Access to NIS+ is based on the same general concepts as the Unix file system model. There are a fixed number of access rights which are granted or denied to the owner, the group, and the world.

Every piece of data (an NIS+ object or an entry in a table) has access rights associated with it. For NIS+ data there are four types of access: create, destroy, modify, and read. Each of these types of access is applied to the population's owner, group, and world.



Read access lets you resolve the object and read its value. Create lets you add new objects to the namespace controlled by the current object. Destroy lets you destroy parts of the namespace controlled by the current object. Modify applies to the contents of the object and not to the object itself (you can change the value but not destroy it).

Tables provide access rights to entries on a column-by-column basis. This means that a particular user might be able to read one column but not the other. If a user does not have read access on a column, NIS+ will mask it out (not return any values).

Access rights indicate whether a user can do something with an object. Before granting access rights, however, the user must be identified. In NIS+ the concept of authentication applies to principals: a more general concept than a user since it also includes hosts and programs.

Authentication in NIS+ is based on Secure RPC, which uses a public key cryptosystem. Security is discussed in more detail in Chapter 10, but the basic idea is to give each principal a set of credentials. The credentials table has three pieces to it:

- A principal's name
- A public key
- A private key

The private key is encrypted using the principal's password. With the public key of a user, data can be encrypted so that only that user can read it: only the private key can decrypt something encrypted with the public key.

The reverse is also true: if the principal encrypts something with the private key, which is a secret, we can verify that it was in fact the principal by decrypting the message successfully with the public key.

To retrieve the credentials for a user, we use the name column as a column that can be searched and refer to an indexed name:

```
(name=principal_name),cred.principal_directory
```

In NIS+, two principals can thus authenticate themselves to each other by retrieving the credentials. Once the two servers have verified that they can work together, they exchange a symmetric key (significantly faster than the public/private key system) for use in future transactions.

This system will even work with a diskless workstation, providing greatly increased security for remote booting. The diskless machine's private key is stored in an EEPROM in encrypted form. A server is able to verify that this is a valid client by comparing information encrypted with the workstation's private key with the public key stored on the server.

To verify that a server is valid, each machine has an NIS+ coldstart file that has the network address of a trusted name server and the server's pub-



lic key, allowing it to verify that any messages from the server are in fact from that server and not some other system masquerading as the server. This system works quite well for two systems to perform mutual authentication, but there needs to be a way to verify another form of principal: the user.

Unfortunately, users do not have EEPROMs (although they may have smart cards). Until smart cards are more prevalent, users get authenticated in two steps. First, the user logs into a local system using the login mechanism. Then, that local machine will act as a mediator to the NIS+ server (at this point the local machine is already trusted). The local machine retrieves the user's credentials from the name server and decrypts it using the user's password. With credentials in hand, the user can now be authenticated to the name server.

The result of this are secure credentials that the machine keeps on behalf of each user. When logging off a machine, the user can instruct it to destroy any credentials it is holding. This authentication process is performed once for each association of principal to principal, until a credential expires.

When a credential expires, authentication must be repeated. Trolling the namespace, as in the case when many referrals are returned for searches, also causes a repeated series of authentication. This can be slow by comparison to the actual lookup time for an object value but should be fairly infrequent.

### *Replication of the Namespace*

Every object has a primary service point, which is where updates occur. It ensures that updates happen atomically and that they are serialized. After the update takes effect, the primary service point is able to propagate the object to secondary service points.

As with NIS, a primary service point (a master) notifies the secondaries (slaves) that an update has taken place. It is then up to the secondary to request the update. This means that a secondary can decide how often to actually get the data (e.g., every 10 minutes, every three updates, after 10 seconds, or any other policy).

Updates between a primary and a secondary are done over TCP. Only a complete transaction causes a secondary to update its replica. A transaction is a single RPC that completes from request to response.

If a primary is lost, updates cannot be performed. However, it is possible to reassign the primary service point. This is done by a principal that has modify access rights to the parent directory of the server in question.

In NIS, the "truth" for data is in the source text files. The NIS map is a version of the truth: if `makedbm` was not run, the map is out of date. In NIS+, the database is the truth. Any text files are just dumps of the database.

Since there are multiple copies of the database, we have a “truest” version of the truth: the primary service point. However, all the other copies are pretty close. If the primary service point goes, it is not a big loss of data. Only unpropagated updates are lost instead of the whole map.

### *Backward Compatibility*

One of the basic design assumptions in NIS+ was to make sure that existing NIS clients could continue to operate in an upgraded environment without change. This means that an NIS client is able to interact with an NIS+ server without knowing it.

In NIS+, the NIS domain is a local directory. An NIS map is an NIS+ table with entries of two columns, one of which can be searched. One advantage of NIS+ of course is that dual maps, such as `hosts.byname` and `hosts.byaddr`, can be put into a single map.

An administration tool converts from NIS maps into NIS+ tables. It is important to remember that when NIS+ emulates NIS, there is no security. If a map has no read permissions on it, there will be an error message returned to the NIS user indicating that there is “no such map.”

To keep backward compatibility, most of the NIS client libraries will either continue to issue NIS requests or will convert internally to an NIS+ request.

Just as the bottom of an NIS+ tree may be an NIS-administered domain, the top is often a Domain Name System service. For example, an NIS+ directory `Foo.Sun.Com` may have a DNS parent `Sun.Com`. An NIS+ request for `Bar.Sun.Com` will return the message:

```
foreign ns == DNS, server == x
```

It may occur to the reader at this point to inquire why DNS is not used as a replacement for NIS instead of adding yet another naming service. One reason is that NIS+ is backward compatible with NIS, allowing current applications to migrate gracefully. More importantly, DNS has several problems that make it an inadequate replacement for NIS. First, security is nonexistent in DNS: a read-only name service with unrestricted access to data. NIS+ requires access rights down to individual tables (at least). Further, DNS, as with NIS, is a read-only system. Administration of data in DNS is done by the transfer of entire zone master files, making update propagation difficult.

DNS limits data to ASCII only (or ASCII-encoded binary). Since there is also a packet size restriction, the relatively large objects which may be needed for a local environment can't really be stored in DNS. Of course, the very limits that make it unsuitable for a local environment also help

ensure wide compatibility and efficient operation in a wide-area environment.

What DNS did do was demonstrate that there can be a hierarchical namespace with separately administered (but cooperating) name servers. This concept is used in NIS+ but is combined with the flexibility needed in a locally-administered environment. “Locally-administered” is a relative term of course, referring to groups of tens or hundreds of thousands of people, in comparison to the millions handled by a system like DNS.

## Domain Name System

While Sun was trying to solve the problem of distributing system administration information through NIS, the Internet had a different kind of problem to solve: finding hosts, services, and people in a wide-area network with explosive growth.

Before DNS, the ARPANET used a single file called `hosts.txt`. This file had a list of every host, and FTP was used to move the file around the network. The `hosts.txt` approach began to have severe problems. As the number of nodes in the network grew, the FTP load on the Network Information Center (NIC) server grew to be immense as thousands of requests for the same file came in. More importantly, adding a new host took an incredible amount of bandwidth on the network. Adding a host meant that `hosts.txt` got bigger. Every other host needed a copy of the file, so that consumption of network bandwidth was increasing by the square of the number of hosts added. There was no problem with 2 or 20 or 200 (or even 2000) hosts, but with networks of millions of nodes becoming increasingly real, the Internet decided that a more decentralized approach would be needed.

Another problem with `hosts.txt` is implied by the name: it only kept track of hosts. What about services? What should be done when a service moves to a new machine? What about people? How is electronic mail sent to a person? Which machine is handling mail for a given population?

Out of this need came a hierarchical, distributed namespace with local caching to improve performance. A name server is considered to be an authority on a portion of the namespace. There may be (in fact must be) multiple servers to provide redundancy and improved performance.

Local caching means that other servers may keep nonauthoritative versions of the data, helping to resolve often-asked queries without continually asking the authoritative server, which may be located many hops away.

The approach in DNS is iterative. If a server can't handle a query, it sends back a referral to the client. The client then sends the request to the server in the referral, which may result in yet another referral. Eventually, the client gets to a server who can handle the question, and it sends back an answer.



It is possible to have recursive lookup in DNS, as an option. In this method, a server will contact the next server until it has an answer and then report back to the client.

Information in DNS originates in master files, text files that can be read by a local name server. If a server has to reboot, it simply rereads the master file. Master files are exchanged by FTP, mail, or some other out-of-band facility.

## Servers

In a Sun environment, the name server is a daemon called `in.named`. The Sun implementation of DNS is based on the Berkeley Internet Name Domain (BIND) service. On the client side, there is a resolver. This is a part of an application (located in a library).

A name server maintains two kinds of data:

- Authoritative information, which refers to information maintained in zones that the server in question takes care of
- Cached data, which is obtained through a local resolver (which in turn obtains it from some other server)

Cached data is not authoritative and may be incomplete or out of date. There is a timeout mechanism that ensures that eventually the cached data will be dumped.

## *The DNS Namespace*

As with NIS+ names, DNS names are a series of labels, forming a namespace in a hierarchy (see Fig. 9-9). Names are written from bottom to top:

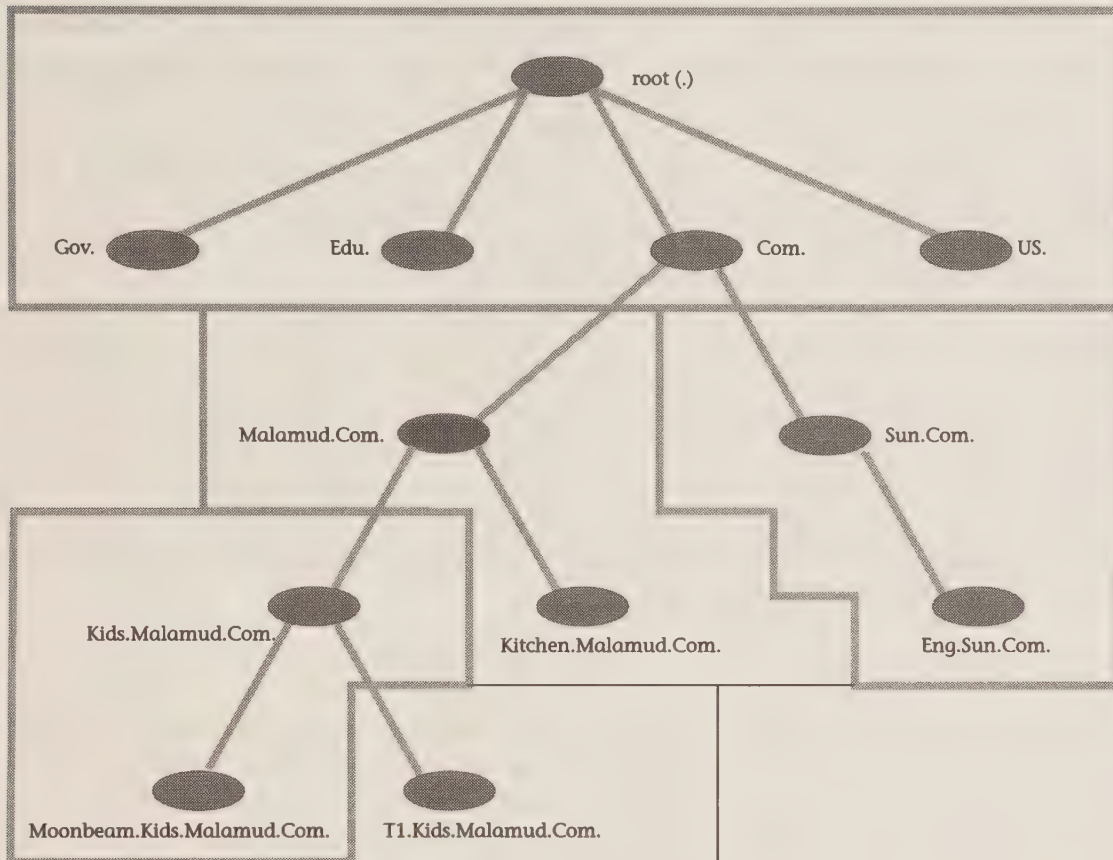
Kitchen.Malamud.Com

This name means the kitchen subdomain of the malamud subdomain of the commercial domain. The root of this hierarchy is maintained by the NIC, which then delegates authority to other servers to handle lower parts of the hierarchy.

Notice that DNS has an explicit root to the hierarchy, including a provision for an organization to administer the root. In NIS+, by contrast, the root is simply the top of the tree. This is because NIS+ is software that is applied to a variety of different namespaces. One of the goals of DNS is to not only provide software but to provide a means for the administration of that namespace.

The Internet has a series of top-level domains administered by the NIC. Edu is used for educational institutions, Gov for government, and Com for commercial. There is also a US domain, which allows U.S. addresses to conform to the geographical naming scheme used in most other countries.





## 9-9 DNS Zones

There are a few organizations that register under the US domain. For example, the administrative home for the Internet, the Corporation for National Research Initiatives, has the domain name:

CNRI.Reston.VA.US

There are no real hard and fast rules as to which domain a group registers in. The author of this book, for example, has registered as Malamud.Com although one could make a very strong case that individuals should not be allowed to be top-level domains and the proper domain name should be:

Malamud.SanFrancisco.CA.US

In addition to the basic domains, there are a few miscellaneous top-level domains. Org is a catch-all used by organizations that don't fit into other categories. Mitre, a research corporation, is registered as Mitre.Org. The

ARPA domain was a transition method used to move old host addresses into the naming scheme.

As with the NIS+ scheme, DNS names represent a string of subdomains, with some object at the bottom. We will see that a variety of objects can be named, although hosts and mailboxes are the two primary uses. Since the system is extensible, users can add other kinds of objects.

The domain name of a node is the list of labels aligned with the least specific label on the right. The root is a null string for a label. A name is represented internally as a length byte plus a string of octets. A zero-length byte signifies the end (root) has been reached.

The external representation of a name skips the length byte and instead separates the names by dots. A dot is specified at the very end as the root (hence, the fully qualified domain name). Relative names are ones without a dot: local software would know what to add. Relative names are either relative to a well-known origin (the default) or come from a search list.

The total number of octets in a full domain name (sum of label octets and label lengths) is 255. You could do things like start with leading digits, but it is recommended that you don't. Case is preserved for storage but is insignificant for comparisons and lookups.

### *Zones*

Within the DNS hierarchy is the concept of zones. A zone is any contiguous portion of the namespace. For example, one zone might be the top-level Com. A set of servers are considered authoritative for that zone.

Underneath that zone is a zone called Sun, referred to as Sun.Com. The server for sun.com might be responsible for a large part of the domain tree underneath it, including:

```
Marketing.Sun.Com  
Engineering.Sun.Com  
Hyperbole.Marketing.Sun.Com  
Whitepapers.Marketing.Sun.Com
```

Notice that the zone extends farther down the marketing part of the tree than it does the engineering part of the tree. There is no reason why engineering couldn't have lower subdomains if they choose to:

```
FDDI.Engineering.Sun.Com
```

Every time a resolver needs an authoritative answer to a name query, it has to find the server that is handling that zone of the namespace. Splitting up into zones is a method of delegating authority to lower portions of the name hierarchy.

## *Implementation*

In a Sun network, there are four kinds of servers:

- Primary master
- Secondary master
- Caching-only server
- Forwarding server

Master name servers, of which one is the primary master, are those that contain authoritative answers to queries for a zone. A requirement of DNS is that all zones must have at least two master servers. In fact, to register a new second-level domain with the NIC, the organization must give the names of the servers that will serve that domain.

Of the master servers for a zone, one is named the primary master. All updates to DNS zone files are made on the primary master. One or more other servers are designated as secondary masters. Think of these as the equivalent of NIS masters and slaves.

A frequent configuration is for one zone's primary master to act as another zone's secondary master. In this way, the requirement of multiple servers per zone is satisfied. As with NIS, secondary servers periodically check with the primary to see if there have been any changes to the data.

The other two types of servers are not considered to be authoritative. The caching-only server keeps a cache of information. Eventually the cache will grow stale and will need to be refreshed from an authoritative server.

All servers are caching servers in that they remember queries that have been previously resolved. The caching-only server is a special case of this because it has no authoritative information.

The fourth type of server, the forwarding server, simply forwards all queries to other name servers. A list of server addresses is kept for forwarding purposes.

A special case is the forwarding slave: it always sends queries to a particular master, which in turn takes care of resolving the query. This is the case when you have a local network and want a single point of communication to the Internet to resolve names. Why do this? The real resolver can have a huge cache—something that a workstation couldn't handle.

Finally, there is the resolver, which is simply a library that knows how to contact a name server and interpret the results. All servers have resolvers as a way of contacting other servers. A client is simply a machine that has only a resolver and is not running the named daemon.

## *Configuration*

When a Sun workstation starts up, it becomes a DNS resolver. All resolvers (which includes all servers) use the `/etc/resolv.conf` file, which contains a

```
root# cat /etc/named.boot

; named.boot file for Primary master

; type    domain        source file or host
; First line is the home directory for the name server
; (this lets subsequent files to have relative path names).
directory /var/named

; The way to find the root servers is via a cache file.
cache                                named.ca

; This server is the primary server for this zone
; "puhosts" is the name of the zone file for this zone
primary Podunk.edu  puhosts

; This line says I'm the primary server for this domain
; (which is the same as above)
primary 32.128.in-addr.arpa  puhosts.rev

; This host is the primary server for the local host
; loopback, data to be found in named.local
primary 0.0.127.in-addr.arpa  named.local
```

#### 9-10 The named.boot File

list of server addresses to consult. Note that keeping bootstrap addresses solves a problem that we saw in NIS: the use of broadcast to locate a server, meaning that there needed to be one server per subnetwork. Both NIS+ and DNS allow servers to be very widely distributed. Using an IP address, stored in a bootstrap file, allows an organization considerable flexibility in deciding where to put servers. Of course, in the Internet world, a server may well be located across the country.

In addition to the `resolv.conf` file, there is a `named.boot` file (also in the `/etc` directory) that configures the `named` daemon. This file establishes a particular server as primary, cache-only, or forwarding. It also contains the location of other servers and files.

Figure 9-10 shows an example of the `named.boot` file for a primary server. This same file can also contain information that designates the server to be a secondary file:

```
secondary                                podunk.edu  128.32.0.4 128.32.0.10 puhosts.zone
```



```

root# cat named.ca
; Initial cache data of root domain servers
; list of servers:

```

```

          99999999      IN      NS      NIC.DDN.MIL.
          99999999      IN      NS      A.ISI.EDU.
          99999999      IN      NS      TERP.UMD.EDU.
          99999999      IN      NS      C.NYSER.NET.
          99999999      IN      NS      AIS,BRK,NUK.
          99999999      IN      NS      GUNTER-ADAM.AF.MIL.
; and their addresses:
NIC.DDN.MIL.  99999999      IN      A      10.0.0.51
NIC.DDN.MIL  99999999      IN      A      26.0.0.73
C.NYSER.NET. 99999999      IN      A      192.33.4.12
AOS.BRL.MIL. 99999999      IN      A      128.20.1.2
AOS.BRL.MIL. 99999999      IN      A      192.5.22.82
NS.NASA.GOV. 99999999      IN      A      128.102.16.10
TERP.UMD.EDU. 99999999      IN      A      10.1.0.17
A.ISI.EDU.    99999999      IN      A      26.3.0.103
GUNTER.AF.MIL. 99999999      IN      A      26.1.0.13

```

### 9-11 The named.ca File

The word *secondary* says that this node is a secondary server. The data is the address of a set of servers that have the zone data. The file name says put the data for this zone into this file as a backup. A boot file for a caching-only server simply contains the cache line. The boot file for a forwarding server is a set of addresses:

```

; named.boot file for Forwarding server
forwarders      128.32.0.10 128.32.0.4

```

An important file is the named.ca file, which contains the names of root servers. This information is vital to the functioning of DNS: in order to resolve any name it is necessary to be able to start at the top of the hierarchy and work down. Figure 9-11 shows an example of this file.

### Resource Records

The most important files for servers are the zone data. This is the information that the server keeps on behalf of its clients. The zone data consists of a series of resource records, each one containing a piece of information. Several standard resource record types are defined (see Fig. 9-12) and the system is extensible.

Resource Record	Name	Description
A	Address	Address of a host.
CNAME	Canonical Name	Used for resolving nicknames into real names.
HINFO	Host Information	CPU and operating system type.
MX	Mail Exchange	Indicates the name of a host that will accept mail for the given name, as well as a preference value.
NS	Name Server	The name of a host that is authoritative for a given zone. Typically, a resolver will then want to know the address of that host, contained in an A type of record.
PTR	Pointer	This is similar to the CNAME record but is used for special domains like IN-ADDR.ARPA.
SOA	Start of Authority	Contains the name of a domain, the name of a person responsible for that domain, and configuration information.
WKS	Well-Known Service	This is an Internet-specific resource that contains a list of well-known services provided by a particular host. For each IP service provider (i.e., UDP and TCP), there is a bitmap. The bit map has a bit set at each position corresponding to the ports used by the service.

### 9-12 Information Kept In Resource Records

Each resource record uses a standard format:

```
{name} {ttl} class record_type data
```

A simple example of this would be:

```
SRI-NIC.ARPA. 999 IN A      10.0.0.51
```

This record says that this resource record refers to the name SRI-NIC.ARPA (the NIC's server). The data is valid for 999 seconds and is in the class Internet.

The type of resource record in this case is an Address record, and the data (logically enough) is an Internet Protocol address. Keeping this information means that when a resolver wants to know the internet address of a particular host, the server will attempt to find an A record for that domain name.

The resource records are what a server uses to load up a zone and start answering questions. At the beginning of a zone is a start of authority record, which gives information about a server. An example of this is:

```
@ (      hostmaster.sri-nic.arpa
      45          ; serial
      3600        ; refresh
      600         ; retry
      3600000     ; expire
      86400 )     ; minimum
```

The @ sign means the current origin, which in this case is obviously the root of the tree. The data portion of the resource record indicates the name of the host (SRI-NIC.ARPA) and the name of a person responsible for that zone (hostmaster.sri-nic.arpa). The person is really just an electronic mail address.

The rest of the data gives zone configuration information. Serial is the current version number of the zone file, used to determine if a secondary server has the current data. The refresh time is how long, in seconds, a secondary name server should wait to see if an update is needed.

The retry indicator is how long to wait after a failure. This preserves the host from being inundated with constant repeat requests when the reason for not answering in the first place is that it is already inundated with too many requests.

Finally, there are expiration and minimum indicators for the record. The expiration is when the data should be considered stale. Here it is 3,600,000, which is 42 days. The minimum number is the minimum number of seconds to use for a time-to-live (TTL) value on other records for this zone. The number 86,400 indicates that 1 day is the default TTL.

After the Start of Authority (SOA) record, the zone file will contain a variety of other resource records. In addition, there may be a few special characters. We have already seen the @ sign, which is the current origin. This allows a default that is appended to any partial domain names.

The origin can be set as follows:

```
$ORIGIN engineering.sun.com.
```

In a zone file, a full domain name is one that ends with a period. If it doesn't end with a period, the current origin is appended to it:

```
FDDI
```

would turn into

```
FDDI.Engineering.Sun.Com.
```

Code	Name	Description
<b>Fixed Header</b>		
ID	Query ID	16-bit ID. Copied back in reply to allow a match.
QR	Query Response Bit	1 bit (0 is query).
OPCODE	Query Type	4-bit query type.
		0: standard query
		1: inverse query
		2: server status request
		3–15: reserved
AA	Authoritative Answer	1 bit is set if the responding name server is an authority for the domain name in the question section. Note that the answer section may have multiple owner names because of aliases: the AA bit corresponds to the name which matches the query name or to the first owner name in the answer section.
TC	Truncation	1 bit, which indicates that the length of the answer that is permitted on this transmission channel.
RD	Recursion Desired Bit	
RA	Recursion Available Bit	
Z	Reserved	3 bits reserved: must be zero in query/response.
RCODE	Response Code	4-bit response code.
		0: no error.
		1: format error.
		2: server failure.
		3: name error (Domain name referenced in the query does not exist and this response is coming from an authoritative name server.)
		4: not implemented (e.g., inverse)
		5: refused for policy reasons
		6–15: future use
QDCOUNT	Question Count	Number of entries in the question section (16 bits).
ANCOUNT	Answer Count	Number of RRs in the answer section.
NSCOUNT	NS Count	Number of name server resource records in the authority records section.



Code	Name	Description
ARCOUNT	Additional Resource Count	Number of additional RR listed.
<b>Question Section</b>		
QNAME	Question Name	A domain name (terminated with zero length octet for the null label of the root). No padding on this field, could be odd number of octets.
QTYPE	Query Type	Two-octet code (includes all the type fields, plus the additional codes like * or AXF).
QCLASS	Query Class	Internet, wildcard, or other class.
<b>Answer Sections</b>		All the other three sections are the same: a number of resource records, with the number specified in the COUNT section. Any RR consists of:
NAME		Name this answer refers to.
TYPE		2-octet RR type code.
CLASS		Two octets.
TTL		32-bit unsigned integer.
RDLENGTH		Resource record data section length.
RDATA		The answer.

### 9-13 (Cont.) DNS Message Format

Setting the origin allows a user in a small group to not see the larger organization unless it is needed. Resources thus refer to local resources unless some other domain is specifically mentioned.

## DNS Operation

Zone files define the information that a particular server makes available to resolvers throughout the Internet. This information is made available using a query/response protocol. Each DNS query contains a question, and the reply has zero, one, or many answers.

Figure 9-13 shows the standard format for a DNS message. When a query is received, a few bits are flipped and any answers appended, and the packet is then returned to the calling node. Notice that there can be several different questions in a single query.

Often, a client will only need a particular kind of information, identified by the query type (see Fig. 9-14). In addition, there may be a request for large amounts of information, such as all data for a zone.

Query Type	Number	Description
<b>Queries Based on Resource Records</b>		
A	1	Host address.
NS	2	An authoritative name server.
MD	3	Mail destination (obsolete—use MX).
MF	4	Mail forwarder (obsolete—use MX).
CNAME	5	Canonical name of an alias.
SOA	6	Start of a zone of authority.
MB	7	Mailbox domain name (experimental).
MG	8	Mailbox group member (experimental).
MR	9	Mail rename domain name (experimental).
NULL	10	Experimental.
WKS	11	Well-known service description.
PTR	12	Domain name pointer.
HINFO	13	Host information.
MINFO	14	Mailbox or mail list information.
MX	15	Mail exchange.
TXT	16	Text strings.
<b>Queries Based on Multiple Resource Records</b>		
AXFR	252	Request for transfer of an entire zone.
MAILB	253	Request for mailbox-related records (MB, MG, or MR).
MAILA	254	Request for mail agent RRs (Obsolete—use MX).
*	255	Request for all records.

## 9-14 DNS Queries

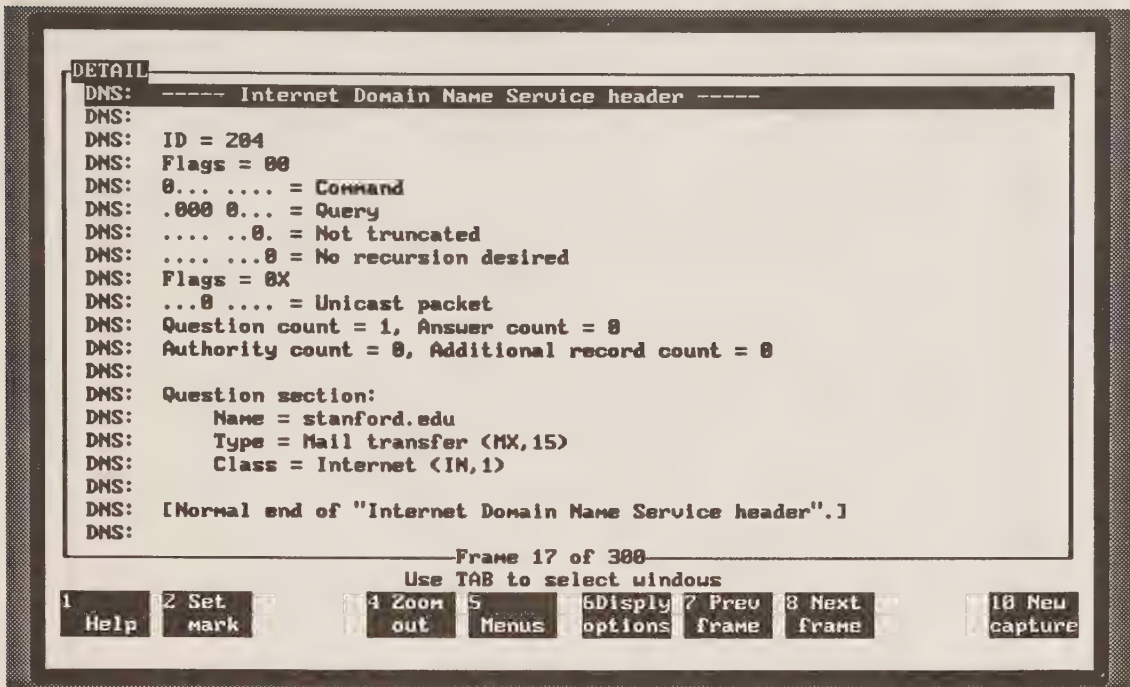
A typical query for an MX record is found in Figure 9-15. The header indicates that there is simply one question, the MX for the domain Stanford.Edu. This will return the name of a host (or the name of a name server to contact since the “no recursion desired” bit is on).

The MX record has two pieces of information for each name:

- Preference
- Host

The preference is a number indicating the priority that this host gets. A particular user can have several MX records. If a name is resolved to several different MX records, the client should start with the one with the lowest value and, if that is unavailable, continue with higher-valued records.

For example, the user Carl Malamud might have the following MX records:



9-15 A DNS Query

Carl.Malamud.Com	MX	0	UUnet.Com
Carl.Malamud.Com	MX	10	OUTHost.Com

This record indicates that to send messages to Carl@Malamud.Com (notice that the @ sign is replaced with a period when the data is put into a zone file), the mail system should first try UUnet.Com. This doesn't necessarily mean that Carl@Malamud.Com has a mailbox on UUnet, only that the host UUnet will know what to do with messages.

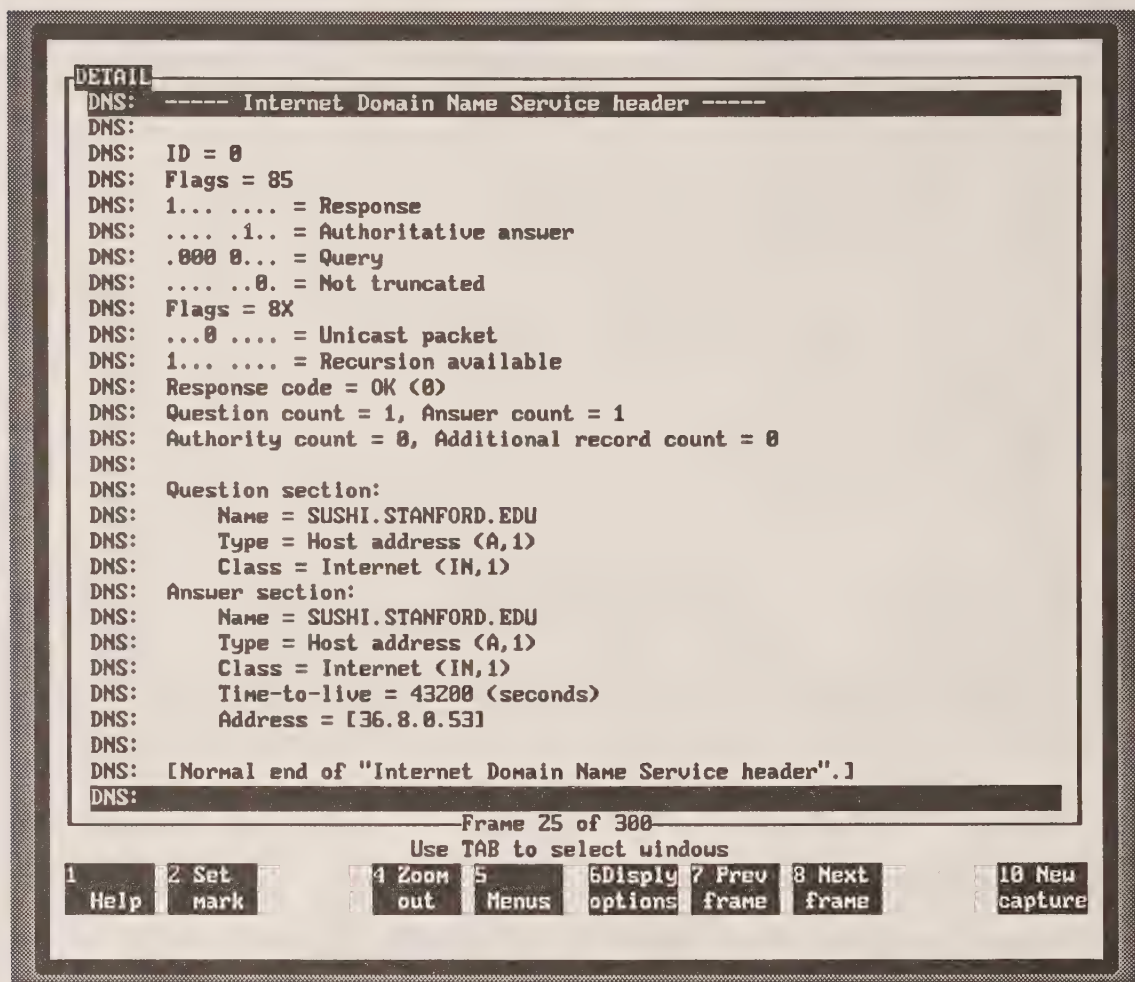
It is possible for a given domain to have many different users. Putting an asterisk into the MX record would match any subdomains:

*.Malamud.Com	MX	0	UUnet.COM
---------------	----	---	-----------

This record indicates that all subdomains of Malamud.Com are being handled by UUNET.COM.

The answer to another query is shown in Figure 9-16. In this packet, the response and authoritative answer bits are set. The answer section contains the internet address for the host Sushi.Stanford.Edu. The TTL on this host is fairly long, 12 hours. Our local resolver will be able to cache this information for 12 hours before repeating this particular query.





9-16 A DNS Answer

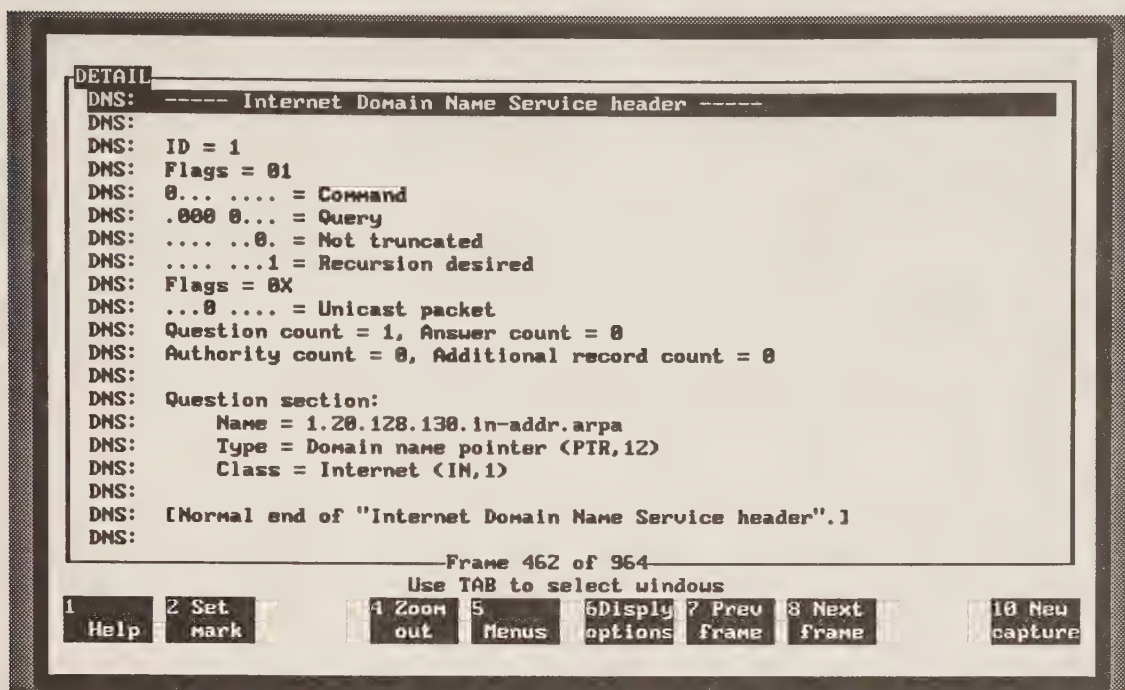
### *The IN-ADDR.ARPA Domain*

Finding hosts and mailboxes are important functions of DNS, but it has one more important function: finding networks and the gateways that might handle the networks. This is not the same as dynamic routing protocols but does form a static bootstrap function.

The basic idea is, given an IP network address, to find the gateway responsible for that network. This was handled by inventing an artificial sub-domain called Inverse Address and placing it in the ARPA top-level domain. By attaching an internet address to the IN-ADDR.ARPA domain, we have specified the IP address of a network using the DNS syntax.

Let's start with an easy example: a host. A host is like a network address except that all 32 bits of the address space are used. We reverse the four





9-17 The in-addr.ARPA Query

digits of the IP address (so that we go from left to right as DNS requires) and form the following resource record:

```
10.0.0.10.IN-ADDR.ARPA PTR    SRI-NIC.ARPA.
18.0.0.26.IN-ADDR.ARPA PTR    SRI-NIC.ARPA.
```

The host SRI-NIC.ARPA has two internet addresses:

```
10.0.0.51
18.0.0.73
```

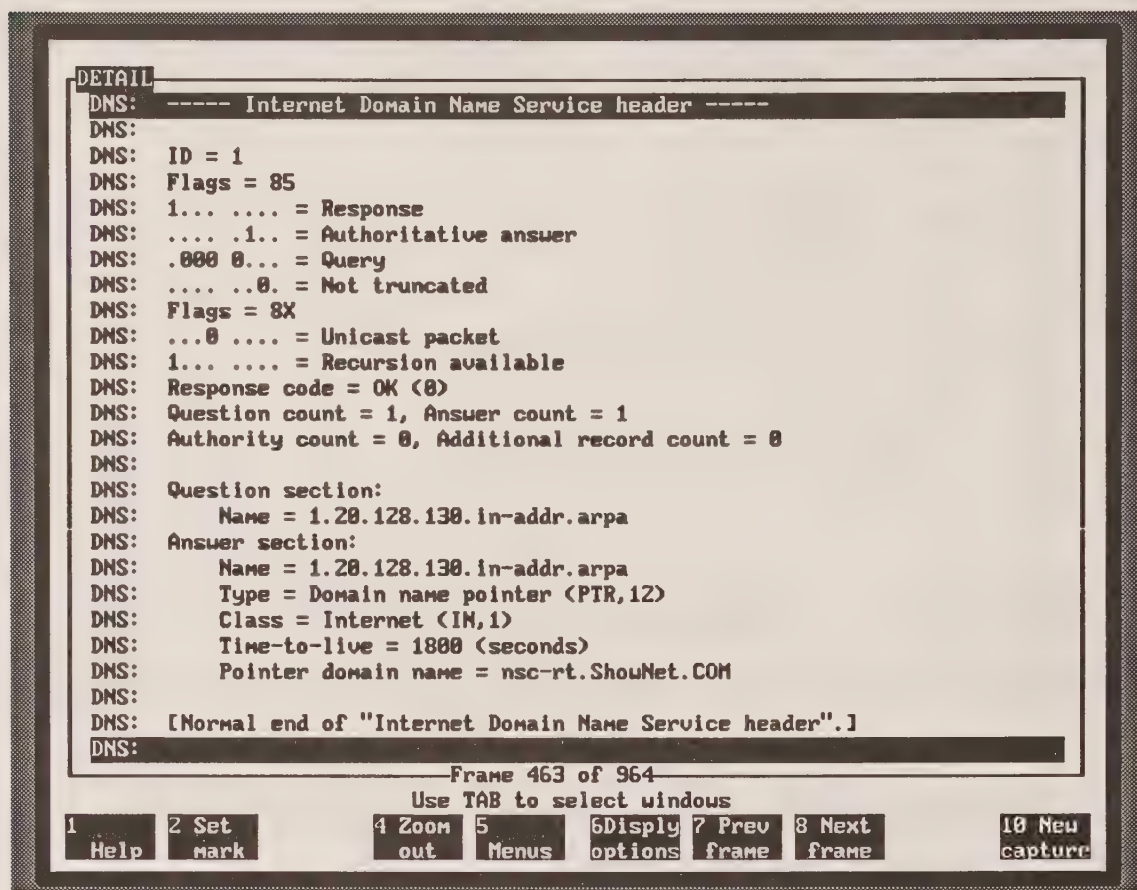
Given a request to open a port that has this source IP address, we can issue a DNS query and find out the name of the host.

If we want to refer to a network, we simply use fewer portions of the IP address space:

```
10.IN-ADDR.ARPA PTR    GW.CSL.SRI.COM.
18.128. PTR    GW.CSL.SRI.COM.
33.12.192. PTR    GW.CSL.SRI.COM.
```

Any address starting with 10., 128.18, or 192.12.33 would go to the gateway with the name GW.CSL.SRI.COM.

In Figures 9-17 and 9-18 we see an example of this type of operation. The question, shown in Figure 9-17, asks for the name of the host that corre-



### 9-18 Return of a PTR Record

sponds to internet address 130.128.20.1 (remember that the address is inverted). The response shows that the host is named nsc-rt.ShowNet.Com, the address of a host at a trade show. Since trade shows are known for volatile networks, notice that the TTL field is extremely short—30 minutes instead of 12 hours.

### Recursion

Recursion asks a server to become a resolver and ask another server a question. This allows the client to be unaware of the actual location of a particular answer. In other words, the workload has been shifted from a client to the server.

To use recursion, both the client and the server must support it. Agreement is negotiated through the use of 2 bits in query/response messages:

- Recursion Available (RA)
- Recursion Desired (RD)

A server will always set the RA bit if it can provide that service. For a client, a query is then submitted with the RD bit set. If a client sets the RD bit, it can only depend on this service if it has received a previous message from that server which had the recursion available bit set. To see if recursion was in fact applied, the answer should have both the RA and RD bits set.

It is, of course, crucial that a server never do recursion unless specifically asked. The reason for this is to prevent endless loops. If there are loops in the name scheme (e.g., a server says another server is authoritative; that server responds with the answer that the original is authoritative) and recursion is automatic, the loop will never break.

### *Caching*

Resolvers try to make extensive use of caches to eliminate network delay, congestion, and server overload. A problem with any cache is that the data may be out of date. Since the authoritative server has no idea which resolvers have cached data, it makes no attempt to notify users of changes in data. Instead, the time-to-live (TTL) indicator is used to control how long data should be used.

Often, a resolver will attempt to use cached data. If that data provides an error, the resolver will then directly contact the authoritative server to get current information. Often, the process of trying to use incorrect information will flush the cache.

The resolver can optionally cache a negative result with a TTL. A typical example is when a name doesn't exist. This is done by adding the SOA RR to an answer if you are authoritatively answering. If so, the minimum TTL in the SOA is the one that should be used to do the negative cache.

### *Message Compression*

Labels in DNS messages are represented by a length indicator and the actual data. What if you need to refer to a very long domain name over and over and over again? A series of labels at the end of a domain name can be replaced with a pointer, which is a two-octet sequence with the first two bits set.

Setting bits 1 and 2 helps distinguish a pointer from a label (labels must begin with 2 zero bits because they are limited to 63 octets in length). The offset is from the start of the message (the first octet of the ID field in the domain header). What this means is that there can be three types of domain names in a message:

- Series of labels ending in zero octet
- Pointer
- Series of labels ending with a pointer



The local resolver will, of course, merge these three types of names before presenting them to the user.

### Combining Name Services

We've looked at three name services: NIS, NIS+, and DNS. There are at least two other important name services: X.500 and the OSF/DEC Naming Service.

X.500 handles the same types of issues that are addressed by DNS, but does so in a much more complete (and thus more complex) manner. X.500 directories can be searched by attribute values and types, and they support a much richer hierarchical naming scheme.

X.500 is used frequently in electronic mail environments, as in the example of the White Pages prototype running on the PSI commercial network and in other environments. It is not widely expected that X.500 will have enough performance to handle the rapid lookups in a local environment.

DEC's Naming Service is part of their DECnet Phase V architecture and has also been adopted by the Open Software Foundation (OSF). NIS+ and DEC's proprietary system both provide name resolution services within the confines of an organization's network. The hierarchical nature makes them both suitable for corporation-wide administration of a namespace.

Many commentators view the services as complementary. X.500 tomorrow and DNS today act as the glue to provide pointers to the appropriate machines at the edge of an organization's network: pointers to mail hubs, routing gateways, and the like.

Once an environment has been reached, the local naming service takes over. Since the local naming service is confined to the boundary of a group of equipment or an organization, it doesn't have to be quite as general. This allows more sophisticated operations, such as the NIS+ table lookups, to occur.

The important point to understand here is that there are two problems. One is the administration of names. This is an issue that any organization is going to have to handle, and it is much more difficult than the second part: the software that implements the namespace.

### For Further Reading

Malamud, *Analyzing DECnet/OSI Phase V*, Van Nostrand Reinhold (New York, 1991). Contains a discussion of the DNA Naming Service.

P. Mockapetris, *DNS Encoding of Network Names and Other Types*, RFC 1101, April, 1989

———, *Domain Names - Concepts and Facilities*, RFC 1034, November, 1987.



———, *Domain Names - Implementation and Specification*, RFC 1035, November, 1987.

Sun Microsystems, *System & Network Administration*, Part Number: 800-3805-10, Revision A of 27 March, 1990. Contains information on NFS, NIS, and other general administration issues.



# Security





# Security

## Secure Networks?

Security in most Internet applications is somewhat suboptimal: the user types in a password to use services such as Telnet and FTP. The password is simply placed into a packet and sent to the remote site. There is a reason for this: many Internet applications are meant to increase access of remote resources, not to control access. This is not to say that Unix systems and TCP/IP networks cannot be made secure (or at least more secure). There are a variety of options that can be used, starting with fiber optic media, control over routing, session-level authentication, access control with auditing, and finally, encryption of the actual text exchanged by two applications. Needless to say, we will not look at each of these methods in depth. Instead, we will look at the unsecured world and will then look at two of the more important security mechanisms.

Clear text passwords on the network have the characteristic of being highly susceptible to a passive attack: a user can simply listen on the network. Most workstations allow their Ethernet adapter to go into promiscuous mode, thereby absorbing all (or most) packets on the network. Simple utilities can then scan the packets looking for initial logon packets and pull out the passwords.

Clear text passwords are fine in two situations:

- A trusted environment
- An environment where security doesn't matter

A trusted environment could be a local workgroup. If the area and network are physically secured from unauthorized access, clear text passwords may not be a big issue. Likewise, if the system being accessed is a simple public document server, the aim may be simply to prevent sabotage, not access.

Passwords only solve one part of the security puzzle—authenticating a remote entity. They are a crude way of determining that a person who typed in a particular username is in fact the person associated with that username. Of course, if you lose your password (or somebody guesses it), the authentication method has been circumvented.

Once the person has been authenticated, there is a further problem of authorization—determining what the person can do. Some systems have no access control: the Domain Name System, for example, allows any user to examine any data. Just because DNS has no access control (or authorization for that matter) doesn't mean that the service serves no purpose: one simply doesn't publish private information on DNS.

Finally, there is the issue of accountability. If a person has been authenticated and is then given a certain level of authorization, it might be nice to know if that person is spending all day trying to break into files. Knowing what people are doing on a system provides accountability (subject, of course, to the privacy rights of the individual user on the system).

There are a variety of tools to provide these basic security services. What is key is that security is not at one place. Rather, there are a series of fences that can be erected around a network to provide varying levels of security and different checkpoints. For example, a security service at the RPC level of the network provides a broad range of security services. This is one set of fences that an intruder must try to overcome. In addition, most operating systems erect another fence around the kernel: no matter what the RPC server will let you do, the kernel still makes an independent assessment about which resources can be used.

This chapter starts with a look at the basic unsecure security that is built into common Unix and Internet services. Then, the basic RPC security flavors, each of which allows a differing level of protection, are explained.

As security reaches higher levels of protection, it is typically based on two forms of cryptography: public key encryption and symmetric key systems. Both of these and their relation to the Sun security environment will be examined.

## Basic Security

Clear text passwords over the network are not only unsafe, but they are a nuisance. People get tired of typing in passwords over and over, so they write little command files that have the passwords embedded in them.

Think about this for a minute. Chances are if you're too lazy to type in passwords, you also have a bad memory. So, you tape the password under your keyboard on your workstation. In your home directory, you've got a few little Unix scripts called `node1.login`, `node2.login`, and `node3.login`.

At least three security problems should be apparent here. First, the workstation is not physically secure, so anybody could walk up to the workstation and reboot it. Second, a smart hacker would of course check under the keyboard and find the password. Third, and worst of all, once the hacker has made it onto the home account on the workstation, batch files give that person access to three other machines.

The Unix hosts database helps to cure some of the problems with passwords. The `/etc/hosts.equiv` and `.rhosts` files on a Unix system allow a remote user to access files—not a bad idea when there are lots of accounts on lots of different machines. To get to these accounts, you don't even need the password, you simply “rlogin” to the account.

This loose level of security is not necessarily bad. Rather than have passwords floating around the network, the outermost host acts as a firewall, performing authorization of users. The outermost hosts are the ones that, within a workgroup, the user first logs into: the Internet gateway, a dial-in system, or a workstation. If all the hosts on the network are trusted to perform authorization, internal network security in a workgroup can be a bit loose. On the other hand, it is possible that a request can be cooked up that comes from a host masquerading as a host in the `hosts.equiv` database.

In this environment, we want to make sure that the passwords used to login to the outermost host are as secure as possible. There are tools that can help tighten things up. For example, Sun offers security enhancement to force passwords to be changed regularly and to be more difficult to crack by picking odd combinations.

## Security and RPC

RPC provides a basic authentication service to its application. Four flavors are currently defined (and more can easily be added):

- None
- Unix
- Secure (DES)
- Kerberos

None provides the level of service that its name implies: nothing. A slightly more secure service is the Unix flavor, which is based on Unix-like principles. Secure RPC is based on a combination of DES symmetric encryption and public key encryption methods. Kerberos RPC, the fourth option, is an adaption of the MIT-developed Kerberos service.

Let's start with Unix (since a description of the none flavor is somewhat rudimentary). There are two pieces of code needed for an application to use Unix authentication: the client needs to create some credentials and the server needs to use them.



Creating Unix-style credentials is fairly simple. The client code simply includes a call to the RPC library to the `authunix_create_default` call. This creates a set of credentials that include the caller's user ID (UID), the group ID (GID), a group membership list, and the host name of the client.

The client then takes these credentials and places them in a cookie (an opaque data structure). All RPC calls then include this cookie. The server, assuming it cares, can then place a call to retrieve the credentials and check them. How the server enforces access policies based on the identity of the user is up to the application.

Note that authentication has two pieces: credentials and a verifier. A determined infiltrator could simply cook up a set of credentials, and there is no way to verify that they are correct. In the case of a root user, this is especially easy (you just switch your identity to that of the user you wish to impersonate).

NFS tries to handle this problem by checking the source internet address of a mount request as a verifier of the host name field (and accepting requests only from privileged internet ports). It is, of course, possible to circumvent that by cooking up fake fields, but this attack requires a fairly determined effort.

The Unix authentication flavor is thus appropriate for a fairly trusted environment. It is possible to spoof this mechanism, but in a fairly trusted environment this is not a strong issue.

Aside from its lack of strength, Unix authentication has another problem: its name. Operating systems like MVS, DOS, and VMS all use different schemes. It becomes fairly difficult to map their security schemes into a Unix-centric world. Again, if the environment is a group of Unix workstations in a trusted environment, this method works fine.

## Secure RPC

Secure RPC is a significantly stronger authentication flavor. It is based on two types of cryptography:

- Symmetric cryptography
- Public key cryptography

In symmetric cryptography, two (or more) users share a key which is used to encrypt and decrypt messages. A classic example of symmetric cryptography is the Data Encryption Standard (DES). The word *symmetric* is used because both users share the same key.

The problem with symmetric key systems is how to get this key to the users. We will see that Kerberos (an example of a system based exclusively on symmetric keys) uses a trusted key distribution center to accomplish this. Another method for transferring symmetric keys is to encode them



using another form of cryptography: public key cryptography. In public key cryptography, each user has two keys: a private key and a public key. The private key is kept in a safe place and never shown to the world: the public key is given to anybody who wants it.

The two keys are the inverse of each other. Information encrypted with the private key can only be decrypted with the public key and vice versa. If we get a message that purports to be from a particular user, we simply apply the public key to that message. If it successfully decrypts, it must have been from the purported user because the only way the message could have been encrypted like this is with that user's private key.

Now that the world's shortest course in public key cryptography is finished, let's see how this works in the Secure RPC mechanism. First, every user needs a network-wide username. This takes the form of:

```
unix.uid@NIS_subdomain.NIS_domain
```

The subdomain.domain is the name of a particular group on the network, the name being kept by NIS+. For this system to work, we want every user to have a unique UID, a service provided by the naming service.

The "Unix" label on the name makes the naming scheme more general. If we want to extend Secure RPC to a VMS world, we might substitute a VMS-specific name:

```
VMS.<uid,gid>@subdomain.domain
```

For individual users, there is no host name, allowing the user to work on any computer on the network (subject to access control issues). For root users, however, there is a need in the Unix world to be able to identify particular instances of superusers:

```
unix.nodename@NIS_sub_domain.NIS_domain
```

At this point we are ready to authenticate a user. The user has a unique network-wide name and has logged onto one system using some local login mechanism. The user also has a public key/private key mechanism. The public key is kept public, possibly as an NIS+ table attribute.

The private key is kept in the user's file system, encrypted with the user's password as a key. Encryption of the private key is particularly important since the whole security structure assumes that this key is kept hidden.

In this system, it is up to the client to initiate the transaction. The user picks a DES key randomly (or a library picks the key for the user), which is known as a conversation key. This key will be used between the client and the server for this particular session, and is encrypted using the user's private key. Along with the encrypted conversation key, the credentials sent by the client include two more pieces of information: the name of the client and a window. The window indicates how much variation will be allowed

before a timestamp will be rejected, and is encrypted with the conversation key.

In addition to the credentials, the client sends a verifier which is intended to make sure that somebody else didn't record the credentials earlier and is now trying to play back a prior session. The verifier consists of two pieces of information. First, there is the current time encrypted with the conversation key. Second, there is the window value, also encrypted with the conversation key.

The server is able to use this information to extract a conversation key. When the server sends information back to the client, it doesn't need credentials, but it does send a verifier. The verifier is the timestamp previously sent by the client, minus 1 (and, of course, encrypted using the conversation key). Subtracting 1 from the timestamp and sending it back is a way for the client to verify that this is in fact the correct server and not some other entity masquerading as the server.

Future exchanges between the client and server repeat this pattern. The client sends its ID (no need for full-blown credentials anymore) plus a verifier, the time encrypted with the session key. The server responds with the time minus 1.

At this point we have a way of forming credentials and delivering them to the server. The server RPC library will check the credentials to see that they are well formed: e.g., that they decrypt properly. However, the RPC library can't tell the application what to do. It is up to the application to decide what to do. Secure NFS will provide an example of how an application decides what to do.

### *The Diffie-Hellman Method*

Secure RPC is based on a particular variant of public key cryptography, known as the Diffie-Hellman method. The Diffie-Hellman method, and the other major variant, the RSA method, are both based on the fact that the public and private keys are the mathematical inverse of each other. The two keys are factors of some very large number. It is fairly easy to use one key to decrypt a message encrypted with the other, but to find the other key is computationally infeasible, since it requires factoring very large numbers.

In the Diffie-Hellman method, the private and public keys are both 192-bit numbers in which the public key is equal to some well-known constant raised to the power of the secret key. The public key of A is thus equal to the constant raised to the power of the private key of A. For a particular host-client exchange, the other user, B, will also have a private and public key combination.

We derive a common key to be used between users A and B by taking the public key of B and raising it to the power of the private key of A. The

same common key can be found by taking the public key of A and raising it to the power of the private key of B.

User A can find the common key because that user has the private key of A and can easily get the public key of B. Likewise, user B has the private key of B and the public key of user A. Nobody else has both pieces of information, and the common key is thus private.

Why are the two formulas equivalent? Remember that the public key is the constant raised to the power of the private key. The shared key is thus the constant, raised to the power of one private key and then raised again to the power of the other private key.

We know that the public key of A is equivalent to the constant raised to the private key of A. Doing a little juggling with the prior formula, we see that we can substitute the two terms “constant” and “private key of A” to yield “public key of A.” The reverse is true: the common key can be derived by taking the public key of A and raising it to the power of the private key of B.

### *Root Access to NFS*

Normally, root privileges on one computer do not confer like privileges on some other server. When a user who is root on a client mounts a remote hierarchy, root access to that hierarchy is usually denied (the notable exception is a diskless workstation).

For purposes of server access, the root user cannot use user ID 0. Instead, the UID is mapped to some other value. Normally, that value is 65,534, which corresponds to the user nobody. In other words users, as long as they are logged in as root on the client, get the same access rights to the remote file system as the general public (which may well be no rights at all).

It is possible in NFS to allow root on a particular client access as root on the server. To do so, the name of the client host is put into the export list for a given hierarchy:

```
/usr/src -root=samba
```

More than one client host can be specified using the following syntax:

```
/usr/src -root=samba:tango:mambo
```

It is possible, in a very loose environment, to allow superaccess for all client processes with a root UID:

```
/usr/src -anon=0
```

This particular command allows anybody in the world to do anything to a file system: anonymous access is allowed and the user is given the user ID of root.



A more typical example of the anon option is to map to -2, which is the user nobody:

```
/usr/src -anon=-2
```

A slightly more advanced security measure would be to use the concept of privileged ports. On many operating systems, TCP and UDP maintain several source ports which are only available to users with root privileges.

It is possible to make a kernel modification to have NFS check the source ports for incoming requests. Root requests must come from a privileged port if this option is enabled. Doing so prevents the problem of a user cooking up some Unix credentials and thereby accessing data as a root user (but is nevertheless a fairly weak security measure).

### *Secure NFS*

To provide better protection, a version of NFS called Secure NFS can be used instead. Secure NFS is based on the underlying Secure RPC mechanism. To enable Secure NFS, the system needs public and private keys for each user. In addition, file hierarchies must be exported with the secure option and the mount command must specify that this is a secure mount.

Secure NFS (and the underlying Secure RPC) is based on the public key NIS map. This map has three fields:

```
netname public_key : private_key
```

The network name is the unique network name for the user (e.g., `unix.uid@domain`). The public and private keys are the data values.

At this point the astute reader may wonder how private the private key is when it is published with a naming service that has no access control. The answer is, in NIS, the private key is encrypted with the user's password. In NIS+ the private key is also encrypted, but there is an additional layer of authentication and access control to prevent random users from accessing this information.

Setting up a key for a user is fairly simple. Normally, a key is generated by somebody with root privileges by issuing the `newkey` command. The command accepts either the name of a user or the name of a host (since hosts must also be authenticated). The operation can also be performed by the user with the `chkey` command as long as "nobody" is present in the `publickey` map.

Users can then change their keys, but they must enter their passwords when they do so:



```

user_mach% chkey
Generating new key for malamud
Password:
Sending key change request to NFS Server
Done.
user_mach%

```

If security is truly an issue (and it ought to be an issue if people are going through this procedure), there should not be an entry for the user “nobody” in the network-wide namespace. If there is, a user can create a new entry in the publickey database.

As with all NIS maps, entries are not necessarily propagated immediately. The maps are either immediately pushed, or the user is sent home for the evening so that the propagation of updates will have completed by the next morning.

When a user logs in, the login process uses the publickey.byname NIS map to retrieve the private key. The password is used to decrypt the password and keep it in memory.

It is possible to log onto a machine without using a password. This is the case when the user uses the rlogin command from a remote host, and the name of that host is in the local hosts.equiv file. In this case, if Secure RPC is going to be used, the user needs to enter the keylogin command so that the password can be entered and used to decrypt the private key and place it in the keystore file.

Using rlogin to log into a host without a password and then using Secure RPC for authenticated applications is a bit silly. After all, the local host has decided to be secure but is depending on the remote host to properly authenticate its users. This only works in a secure local network.

The private key is only kept in /etc/keystore while the user is logged on. In fact, it is possible, using the keylogout command, to have the private key destroyed before the user logs out.

Once keys are in place, it is necessary to set up an access control policy inside NFS. The secure option specifies that Secure RPC should be used for authentication, and an access command says who can get to the data:

```
/usr/src -secure,access=engineering
```

Likewise, the clients put their secure mounts into the file system table (/etc/fstab) or into Automounter maps. What happens when the system is exported secure and the user forgets to put the “secure” option into the mount command? The user can still access the remote file, but all access are mapped to user “nobody.” Only files allowing access to the general public may be read.

## Kerberos

Kerberos is an authentication service developed at MIT as part of Project Athena. Kerberos is an authentication server: it is a trusted third-party that will vouch for your authenticity to some other computer.

Kerberos has a variety of clients: people and servers. For these clients, Kerberos keeps a database that contains each of their keys, which are used to encrypt data and to decrypt it again.

In addition to Kerberos knowing this key, the client also knows it. Now Kerberos and the client have a shared secret that they can use to encrypt message traffic between them. This is the first of two Kerberos keys: clients (users and servers) can authenticate themselves to Kerberos using the shared key.

The next step is providing a way that two Kerberos clients can communicate with each other. This is done by a session key. Kerberos takes a session key and hands it to each of the two clients. Now the clients have a shared secret and can use it to communicate with each other.

Not all applications will make the same use of Kerberos. It could be that a service doesn't really care who the clients are. A more sensitive application may require authentication at the beginning of the session. An even more secure server might require authentication of each message. To be really secure (and very slow), the message traffic itself can be encrypted using the shared key.

The key is used to encrypt data. The encryption methods can vary, but most systems use one of the modes of the Data Encryption Standard (DES). The encryption method doesn't really matter—Kerberos is more concerned with how to get the keys distributed to clients.

The Kerberos database consists of one record per client. The record has the client name, the key, and the expiration date for the key. Things like the human name, or phone number, are kept by a name service such as Hesiod or NIS+. Hesiod is an open service based on the Domain Name System, but it is enhanced to add some access control and authentication of users. (Hesiod, part of the MIT Athena Project, is not supported by Sun.)

The basic process of securing a service in this environment is for the user of that service to present two pieces of information:

- Credentials (known as a ticket in Kerberos)
- A verifier that the ticket is not stolen (known as an authenticator in Kerberos)

There are three phases to being able to use a service on the network. First, the user needs to establish credentials with Kerberos. Then, the user goes to Kerberos and gets authenticated for a particular service. Finally, the user presents these new credentials to the end server.

Let's assume for the time being that the various principals have already been authenticated to their Kerberos server and concentrate instead on what a session ticket might look like. The session ticket, which allows a client to access a particular application server, has the following components:

- Server name
- Client name
- Client IP address
- Current time
- Lifetime of the ticket
- The random session key

The random session key is only good for this particular server-client pair, and only for this particular session. This whole ticket gets encrypted with the key of the server. When a client presents this information to the server, the server knows that the ticket must have come from Kerberos, since that is the only other place that the server's key is stored.

Why encrypt the ticket with the server's key? This prevents the client from playing with the ticket, changing the lifetime or the IP address, for example.

A ticket is an authenticator. Once authenticated, however, it is necessary to verify that the client has not been replaced by somebody else who is replaying old traffic. A verifier (or authenticator) consists of three pieces: the client name, the timestamp, and the client IP address, all encrypted using the session key.

Tickets are granted by a special server called the ticket-granting server. This application is responsible for providing the client with tickets to be used for talking to other servers, which raises a classic bootstrap problem: how do we get the first ticket to talk to the ticket-granting server?

When users walk up to a workstation, they are prompted for a username. However, instead of consulting some local Unix file, the workstation will send a packet to Kerberos asking for admission to the ticket-granting service for this particular client.

Kerberos will check to make sure that the client is known to it. If so, it generates a special ticket: admission to the ticket-granting service. This ticket tells the ticket-granting service to expect a client coming in soon.

Kerberos then sends the client a packet that is encrypted with the user's password (actually a function of that password). Inside that packet are two pieces of information. First, there is the ticket which is used to get to the ticket-granting service (the same ticket which was previously sent to the service). Second, there is a copy of the session key for use between this particular client and this particular ticket-granting service.



At this point the workstation has two pieces of information. First, it has the username. Second, it has the encrypted packet of information. The workstation then prompts the user for a password and uses the password to decrypt the packet, extracting the ticket-granting ticket and the key to use to encrypt the authenticator.

Notice that the password never entered the network. Notice also that the workstation doesn't store the user password. Instead, it uses the password solely to decrypt the first packet that comes back. All further transactions are based on the random session key used to communicate with the ticket-granting server.

Eventually, the ticket-granting ticket will expire. At that point, the user is once again prompted for a password. Note, however, that the worst somebody eavesdropping on network traffic can do is steal a random session key—even this is exceedingly difficult.

At this point, the client is able to prove its identity, at least for the duration of the ticket, and the user can start requesting other services. To get a ticket, the client sends a request packet encrypted with the special key that it has for the session with the ticket-granting server.

Inside this encrypted password are three pieces of information. First, there is the name of the desired server. Second, there is a copy of the ticket-granting ticket. Third, there is the authenticator: the encryption of the client name, IP address, and the current time.

The ticket-granting server responds with two pieces of information: a copy of the ticket to be given to the application server and a copy of the key to be used. Needless to say, all this information is encrypted with the session key between the client and the ticket-granting server.

At this point the client is ready to request the actual service. It sends the ticket (encrypted with the server's private key) and the authenticator encrypted with the session key. The server decrypts the ticket and makes sure it is valid.

The server now has one more thing to do: check the authenticator. It decrypts the information and makes sure that the time is valid. This of course means that the clocks of the client and the server must be in sync or the server may suspect a replay.

A truly paranoid server can keep track of timestamps received and make sure that they increase monotonically. This is fairly conclusive evidence that the client is truly there. Servers aren't the only ones that might want to be paranoid: the client can request that the server verify its identity.

This optional return verification is done by the server taking the timestamp it receives and adding 1 to it, very similar to the method used in Secure RPC. The timestamp is encrypted with the session key and sent back.



## *Kerberos and RPC*

Kerberos is fairly simple (at least conceptually) to add to the RPC mechanism. A new flavor of authentication is added called AUTH\_KERB. The RPC library handles Kerberos ticket validation: interaction with the key distribution center (KDC) is transparent to RPC application.

For NFS, a new Kerberos option is added as both an export option and a mount option. If the client does not know which flavor of authentication is being used, the mount protocol passes that information back, allowing the process to be transparent to the user.

As with Secure NFS, a Kerberos principal is one of three things:

- A user client: username.instance
- A root client: root.hostname
- A server: nfs.servername

The instance of a username can be null.

As with any authentication service, just because the remote root client has been authenticated doesn't mean that the access control policy necessarily gives that client any services: the client might still be mapped to "nobody."

## *Is Kerberos the Answer?*

No security system is perfect. Secure RPC has some potential problems, as does Kerberos. Secure RPC and Kerberos share many features: both use a symmetric key for the purpose of running a session. It is only the way that this symmetric key is distributed that differs. In Secure RPC, the public key cryptography method is used to distribute symmetric keys. In Kerberos, the Key Distribution Center (the Kerberos center) and the ticket-granting service is used.

## *Administering Security*

The majority of security problems are caused by lax password policies and failure to apply known fixes to security problems.

Most problems are password related. For example, it is possible to use the Trivial File Transfer Protocol (TFTP) to steal password files. The files in Unix are encrypted, but this at least gives the attacker a list of usernames. Of course, if TFTP is confined to a local environment, the password files are available to that group anyway.

Once an attacker has the password file, it can be scanned for extra UID 0 (root) accounts, accounts with no passwords, or new entries to files. Attackers also attempt to exploit accounts without passwords or known pass-

words. The finger protocol gets the account names, and then simple passwords are tried.

A simple solution to the problem of password files is make sure that the outside access uses different files than do people from within a domain. Inside the domain, NIS maps are used for passwords. A machine at the edge of the gateway serves as the entry point to the network. This system has a password file that is used for operations like FTP.

Access from within the network, presumably a more trusted population, is also governed by use of the `.rhosts` and the `/etc/hosts.equiv` files, both files that name certain machines as trusted. If those files contain hosts from outside the domain, make sure they are authorized. It also would be fairly wise to make sure that the two files are not writable by the world, an easy way to add yourself to the list of authorized hosts.

Most problems begin because of open accounts. Sometimes this is just sloppy administration, as when a vendor ships a machine with a default privileged account and the user never changes the password. VMS is especially prone to this problem since there are several different default privileged accounts.

There is another instance that is harder to fix. Anonymous FTP is a convention to allow public access to archives. The convention says that a group maintains an account called `guest` which has no password, allowing the general public access to resources. If anonymous FTP is supported (and it certainly provides a valuable Internet service), it is important to make sure that file permissions, configurations, and other information are properly secured. A separate password file is essential here.

Once an attacker gets on a system, there are a few tricks often attempted. Most of these tricks are easily protected against. A common one is to examine the `cron` and `at` processes, the two processes that are used to schedule programs to be run later.

If any of the files used by these scheduled programs are world-writable, a user can easily change the nature of the job. Since the job runs with a person's identity, possibly even `root`, it allows commands to be run that will allow the intruder back on (or worse yet do something to the data). The key is that all files and programs referenced by `cron` and `at`, including the job files themselves, should not be world-writable.

Having an authorized user unwittingly execute programs is the basic hole most attackers use. For example, many users have their local directory as the first item in a search path. If a version of the `ls` program can be put into a local directory, that program will be executed whenever a user tries to list the files in a directory.

The trojan horse `ls` would do some work (e.g., add a new account) and then actually list the files so the user would not suspect anything is wrong. While the trojan horse is a danger for any user, it is particularly a problem

for root users. Root users should make sure that they use a known, good copy of critical programs like su, login, and Telnet.

A trojan horse will often try to hide information for future use. For example, it may take your password (say as part of an altered login program) and store it in a hidden place. Common hiding places are in the /tmp directory, or in odd file names like "dot dot space space" or "dot dot dot dot" or even in perfectly ordinary file names like .mail.

### *Remote Booting and Secure NFS*

Assume there is a power failure in the middle of the night. All the secret keys that were stored get wiped out, which means that no process can get access to a secure NFS hierarchy. If the root user is around, he or she can type in the password which decrypts the root private key, which allows things to proceed.

Another problem is the single-user boot. Unix workstations usually allow a user to boot the system in a single-user mode, which gives the person root privileges. Why do this? Believe it not, people will sometimes forget the root password, which would make all supervisory privileges inaccessible.

On Sun systems, this single-user boot is only allowed on a physically secure port, such as the console. When the computer is a workstation, however, that console is simply the keyboard. In other words, if you leave a person in your office for an hour, there is time to reboot your workstation in single-user mode, change various files, and then change the workstation back to normal operation.

There are two ways to solve this problem. The C2 security option with SunOS can be activated. This security feature, part of the standard operating system release, requires a password for all boots, and adds features like logging critical operations.

The other way to handle the problem is to change the console port to be unprivileged, requiring a password to login. Doing so, as in the case of the C2 security feature, means that the root user should remember the password. Otherwise, the system is basically inaccessible.

The final problem of booting is the diskless machine. There is no way to make booting a diskless workstation totally secure. It is possible for a devious (very devious, that is) user to impersonate the boot server and boot some devious kernel that makes a record of the user's secret key. The problem here, of course, is that Secure RPC only provides protection after the kernel and the keyserver are running. Before that, there is no way to authenticate the remote bootserver.

Are any of these things really problems? Not in any realistic environment. For example, writing a devious boot server is a task that ranges from very difficult to impossible. Besides, all a user has to do is boot from a local disk if this security issue becomes a problem.



Most security problems can be easily solved: they just take money. Adding Secure RPC takes money because it slows the machines down slightly. Adding a local disk for booting takes money. It simply becomes a question of how much security is worth to an organization.

Even with the best security hardware and software in the world, the best line of defense is the administrator. This person, through good administrative practices and careful tracking of logs and other audit material, can go a long way to prevent and detect intrusions.

### *Is it Secure?*

Secure RPC and Secure NFS certainly provide good initial authentication, subject to a few caveats. First, the weak link in any system (including Kerberos and anything else that requires a human to log in) is the initial password. Once people guess your password, they are you as far the authentication systems are concerned.

More aggressive attacks, however, are certainly prevented with Secure NFS. Authentication is provided (assuming the person's password is secure). Attacks such as eavesdropping won't work since the time is associated with a particular session key and a window is stated in the initial credentials.

It is possible to tap all transactions going to a server and then pull the plug on the server. The server's credentials table will wipe out and the provocateur is able to replay previous transactions. The only problem with this, of course, is that workstations specify windows of 5 to 10 minutes, and it takes longer than that for the server to reboot. Of course, most network administrators put their servers in a secure place so that any security compromises are isolated to a single workstation and a single user.

It is possible to break the Secure RPC mechanism by taking the digital logarithm of a person's public key. Doing so requires a pretty big hammer: a large supercomputer working for a very long time. Secure RPC thus serves quite well to discourage the casual problem maker but would probably not stop the National Security Agency if it was hot on the trail of international spies.

If security is really an issue, then Secure NFS with access control and authentication should be combined with other measures. For example, there are Ethernet-based LANs that use encryption and fiber optic cable. In conjunction with Secure NFS, as long as the network is limited to some secure physical area, this provides an extremely high degree of security.

### *Performance*

Security, like any feature, costs money because it takes computing resources. The cost of security is typically in the encryption operations. DES-



based encryption is fairly rapid. A timestamp is a 64-bit quantity (which happens to be the DES block size). After the initial authentication, each RPC transaction takes four encryption operations:

- Client encrypts the request timestamp
- Server decrypts it
- Server encrypts response timestamp
- Client decrypts it

On a Sun-3 it takes 0.5 milliseconds (ms) to encrypt in hardware, 1.2 ms in software. Therefore we're adding 2 ms in hardware implementations and 5 ms for software implementations. An average NFS round-trip request takes 20 ms. Encryption thus costs 10 percent of transaction time with hardware, 25 percent with software.

Public key encryption is significantly slower than the DES encryption. This is, of course, the reason that public key and symmetric encryption methods are combined in Secure RPC. In order for Secure RPC to work, both the server and the client must calculate a common key.

On a Sun-3, this process takes about one second. The process is then repeated when the server gets the mount request. It thus takes about 2 seconds to get a secure NFS mount going. The 2-second delay on a Sun-3—the delay is significantly less on a Sun-4—is often not noticed by the user. Many NFS mounts take place in the background, allowing the user to read mail, spill coffee, or do any of the other preliminaries before setting down to work.

The 10 to 25 percent performance hit for Secure NFS transactions on a Sun-3 is also not quite as noticeable to the user as it might appear. Users have a mix of local, normal NFS, and secure NFS operations, so the throughput decline would be some weighted average.

### For Further Reading

Denning, Peter J., ed., *Computers Under Attack: Intruders, Worms, and Viruses*. Reading, Mass: Addison Wesley, 1990.

Kent, S. *Privacy enhancement for Internet electronic mail: Part I—Message encipherment and authentication procedures [Draft]*, RFC 1113, Aug. 1989.

———, *Privacy enhancement for Internet electronic mail: Part II—Certificate-based key management [Draft]*, RFC 1114, Aug. 1989.

———, *Privacy enhancement for Internet electronic mail: Part III—Algorithms, modes, and identifiers*, RFC 1115, Aug. 1989.

Security Study Committee (David D. Clark, Chairman), National Research Council, *Computers at Risk: Safe Computing in the Information Age*. Washington, D.C.: National Academy Press, 1991. ISBN 0-309-04388-3, 320 pp. Available in bookstores or by calling (800) 624-6242.



# The Automounter





# The Automounter

## The Problem with /etc/fstab

NFS and NIS provide a very flexible way of finding and mounting remote file systems. However, in very large networks it doesn't make sense to have the user keep all file systems mounted simultaneously, nor does it make sense to have the user keep track of all systems in the network. The Automounter addresses both of these problems, plus has the significant advantage of allowing users to mount file systems transparently, on demand, and without having root privileges.

Before looking at the Automounter, let's look again at an environment that uses just NFS. Every client system has a file system table (/etc/fstab) which contains the location of file systems that need to be mounted: local systems, NFS systems, and others such as AT&T's RFS. The alternative to hardcoding the file systems into /etc/fstab is to use the mount command—not very convenient, as well as requiring root privileges.

Hardcoding remote file systems into /etc/fstab has several problems. First, if the remote system needs to be moved, the network administrator (or even worse, the user) must change all clients. Take a corporation-wide information service, such as a policies database, for example. This database may have been first installed on a Sun-3 with a 200-Mbyte drive. As with many corporations, the number of policies has accelerated. The network administrator now wants to use a Sun-4 with 4 Gbytes of disk drives. In order to do so, we probably have to have all clients change their /etc/fstab, which is not very convenient. On a network with more than 100 nodes, this becomes intolerable.

A second problem with hardcoding remote mounts is that neither the mount command nor the /etc/fstab file can handle replicated file systems. In the case of manual pages, for example, there may be several instances of the file tree throughout the network. It would be nice to name a preferred manual server and to be able to use the others as a backup.

Finally, there is a need to mount file systems on demand. Keeping a file system mounted uses resources. Granted, the stateless nature of the protocol means that the resources required are minimal. However, with hundreds of potential file systems, resource demands begin to add up. Also, it is inevitable that the location of file systems will change over time in a large network (i.e., more than 100 nodes), requiring unmount and remount operations.

### Automounter Overview

The Automounter is a program on the client workstation that emulates an NFS file server, providing a much more flexible file system than can be had with hardcoded mounts. The kernel is thus able to issue requests to the daemon just as it would to any remote file server, using the local host address instead of a remote host address (see Fig. 11-1).

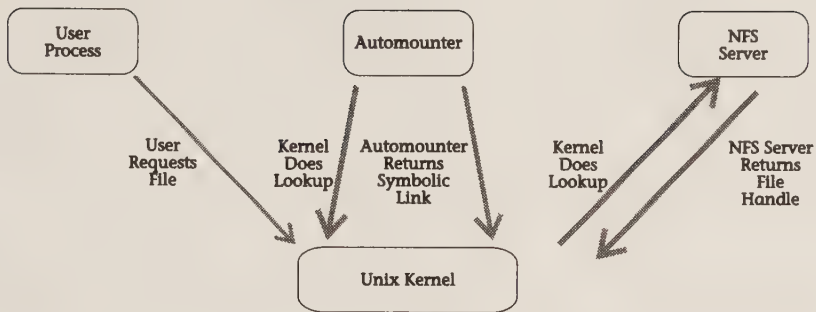
The Automounter registers itself with the client kernel as serving certain mount points, just as the kernel keeps track of other mount points served by the local file system or remote NFS servers. When a request comes in for one of the served mount points, it is handed to the Automounter. The Automounter consults a database that it maintains to see which remote server has the required file system. A real mount request is issued to that remote system, and a reply is sent back to the kernel by the Automounter indicating that the mount point is actually a symbolic link for another file system (the one now mounted) and that it should go there for the data.

Figure 11-2 shows how the Automounter fools the local file system. The user requests access to a directory called `/remote1`. The Automounter has registered itself as the NFS server for that file system. The kernel thus issues a request to the Automounter for `/remote1`. The Automounter looks in the internal database and sees that this file system is exported by a remote server, `server1`.

A mount request is issued to mount `server1:/remote1`. The actual mount point is in a temporary directory called `/tmp_mnt`. The user does not want to be aware of any of these details, so a symbolic link is created between what the user sees, `/remote1`, and the actual file system. The Automounter then returns the symbolic link.

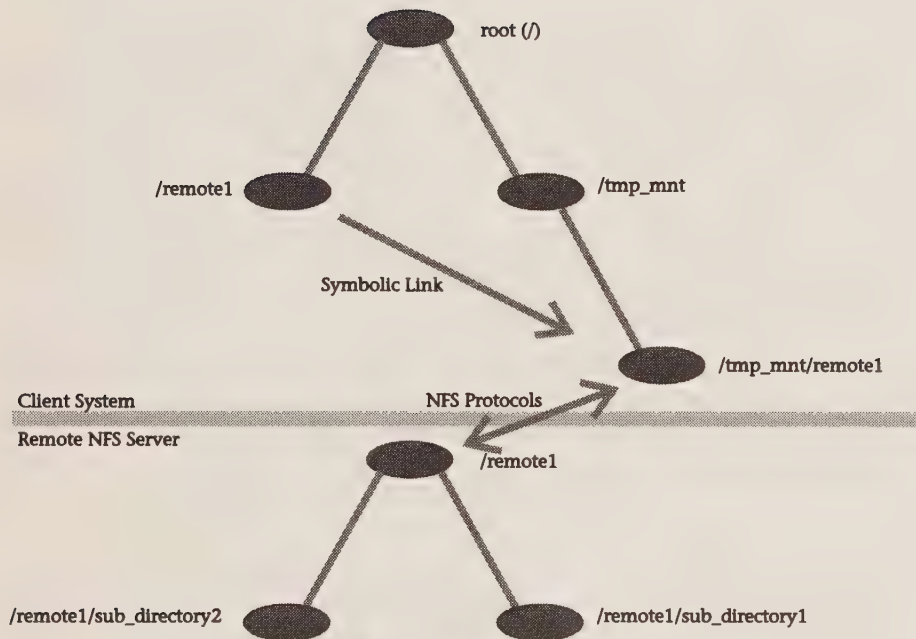
At this point, the kernel will typically follow the symbolic link to `/tmp_mnt/remote1`. That mount point is handled by the remote NFS server, and a series of NFS packets are dispatched to satisfy the user requirement (e.g., to list the files in a directory).

Notice that once the symbolic link is returned, the Automounter is out of the picture. The user requests files, changes to subdirectories, and does all the other normal file operations directly with the remote NFS server.



## 11-1 Automounter Processes

■ ■ ■ ■ ■ ■ ■



## 11-2 Mount Points

At some point, the user will stop accessing this file system. The Automounter monitors access, and after a time interval of inactivity (5 minutes is the default), it unmounts the remote file system. Note that the user still has the option of changing to `/remote1` again. This would cause the Automounter to remount `server:/remote1` and again return a symbolic link.

## Emulating an NFS Server

The Automounter supports a subset of the NFS protocol. Those calls that actually access files (e.g., create, remove, setattr, read, write, and rename) are not needed by the Automounter. Instead, it supports the calls that the kernel would use to locate files and directories:

- null
- lookup
- getattr
- readdir
- statfs
- readlink

Since a user would need to use one of the above calls to locate a file, supporting this subset is enough to fool the kernel into thinking there really is a file system mounted. The result of these calls are symbolic link pointers to another file system.

The Automounter begins operation by opening a UDP loopback socket and registering itself with the local portmapper service as an NFS server port. It then forks off the Automounter service daemon, which starts to listen on the UDP socket for NFS requests.

Meanwhile, the Automounter command consults a local database (a set of NIS maps and local files) and issues a series of mount system calls to the kernel to notify itself about handling certain mount points. Although the mount system call varies according to the kind of file system, the Automounter typically issues a call as follows:

```
mount ( "nfs", "/usr", &nfs_args)
```

In a real call, the `nfs_args` contains the network address of the NFS server. The network address doesn't really have to be an IP address; it can be a local process, which in this case is the automount daemon. We are thus using the loopback version of the RPC protocol to contact the Automounter daemon.

What we've done is said that if you need the following directory (e.g., `/usr`), you should contact the local process, which is the Automounter. It will act like the `/usr` file system, with the important difference being that it



always ends up returning symbolic links to the kernel instead of actual data.

## Maps

Configuration of the Automounter is handled by a series of maps that contain the name of the file system to be mounted and where it should be mounted locally. When the Automounter is started (usually in the `rc.local` file), it is passed the name of a map:

```
automount /src /etc/srcmap
```

This command says to make the `/src` directory a mount point and to use the file `/etc/srcmap` to find out which remote file systems may be accessed from here.

There are three kinds of maps used in the Automounter:

- Files
- NIS maps
- “Built-in” maps

The built-in maps, discussed later, use existing files to provide the illusion of an extension to the local file system. The files and NIS maps allow a combination of network-wide automount configuration (the NIS maps) and local modifications (the files).

The map mentioned in the `automount` command line is the master map. It may in turn point to other kinds of maps. By convention, the master map is contained in the file `/etc/auto.master`.

A master map is basically a list of other maps and contains three fields:

```
mount_point      map_name          ( mount_options )
```

The mount point is the full pathname of some directory. If the directory does not already exist, the Automounter will create it. If the directory does already exist, mounting it will hide its contents.

The maps pointed to by the master map can be direct or indirect. Both have the following fields:

```
key              ( mount_options )    location
```

The only difference between a direct and an indirect is the key: for a direct map the key is a full pathname. For an indirect map it is a simple name.

A typical example of a direct map entry is:

```
/usr/man        -ro,intr          manual:/usr/man
```

This entry tells the Automounter to create a mount point `/usr/man`. When the user references this entry, the Automounter will mount `manual:/usr/man` in read-only mode with interrupts allowed.

A typical example of an indirect map entry is:

```
parsley          -ro,intr          veggies:/usr/greens
```

This entry tells the Automounter that a subdirectory called `parsley` is actually a mount of `veggies:/usr/greens`. So where does `parsley` get put on the local file system? Indirect maps are relative to some mount point. That information is provided either in the command line or in another map. For example, the master map may contain a line that reads:

```
/veggies          /etc/auto.veggies  -ro,soft,nosuid
```

What we've done with this is taken the mount point `/veggies` and then added all the entries in the `auto.veggies` map underneath it. What we end with is:

```
/veggies/parsley  => veggies:/usr/greens
```

At this point we can look at a typical `auto.master` file:

#Mount-point	Map	Mount_options
/-	/etc/auto.direct	-ro,intr
/home	/etc/auto.home	-rw,intr.secure
/net	-hosts	

This master map has three entries in it. The `/-` mount point is really filler: it indicates that full pathnames will be contained in the map called `auto.direct`. The `/home` mount point points to the `auto.home` map. Finally, `/net` points to a special built-in map called "hosts."

### *Indirect Maps*

A typical indirect map is `auto.home`:

```
jones            willow:/home/jones
pynchon          cypress:/home/pynchon
malamud          malamud:/home/malamud
```

Why do this? Let's say by convention, we have made every user's home directory follow this syntax:

```
/home/node_name/user_name
```

Throughout the network, the password database makes every user's home directory follow this syntax. In Unix, if the user refers to a directory `~user-`

name, that directory is expanded to their home directory. For example, user Tom may have a home node of Pynchon. The following commands would work in a local computing environment:

```
othernode% cd ~tom
othernode% pwd
/home/pynchon/tom
```

The change directory command moves the present working directory to be /home/pynchon/tom. The auto.home map allows this to work in a networkwide environment:

```
othernode% cd ~carl
othernode% pwd
/home/malamud/carl
```

The ~carl reference was first expanded to be /home/malamud/carl. Next, the kernel attempts to change to that directory. Since /home is an Automounter mount point, this is intercepted by the Automounter daemon. It consults the auto.home map, mounts the remote file system, and returns a symbolic link.

This naming convention, in conjunction with the Automounter, allows a user to change to any other user's home directory:

```
othernode% cd ~joe
othernode% pwd
/home/joe_node/joe
```

At this point two caveats are in order. For this to work in a networkwide environment, the password database should be administered by NIS so that a user can find a home directory on any node of the network. If NIS is not used in conjunction with the Automounter, the network administrator must distribute and keep consistent local copies of Automounter maps—not much of an improvement over hand-coding /etc/fstab files. NIS and the Automounter should be used together.

The second caveat is that just because the Automounter attempts to do a mount, this doesn't mean it will be successful. User Joe must have exported that file system. Even if the user can change to that directory, that doesn't mean that user Joe has permitted the world to read his files.

## Multiple Mounts and Multiple Locations

Both indirect and direct maps allow an extended syntax that makes the Automounter significantly more flexible than a manual mount. Take the following map:

/usr/local \	
/bin	ivy:/export/local/sun3 \
/share	ivy:/export/local/share \
/src	ivy:/export/local/src

This is basically one long entry made a bit easier to read via the slashes. It is in fact three separate mount points: /usr/local/bin, /usr/local/share, and /usr/local/src. Note that each of the mounts could be a different server and could have different options.

Why do a multiple mount? When one of the areas is referenced by a user, a multiple mount ensures that all three are mounted. If the entries were on three separate lines, they wouldn't be mounted until explicitly referenced. Let's say there are three separate entries:

/usr/local/bin	ivy:/export/local/sun3 \
/usr/local/share	ivy:/export/local/share \
/usr/local/src	ivy:/export/local/src

If a user changes to /usr/local/bin, only one file system has been mounted. At this point, the user might type the following command:

```
mynode% cd ../src
../src: No such file or directory
```

The change directory (cd) command said go to the parent directory (the "..") and then to the subdirectory src. At this point, there is no subdirectory src since it has not been mounted. In a multiple mount, when one subdirectory is referenced, the whole tree is mounted. This means that there is a mount point for each point in the hierarchy.

### *Mount Points and Options*

In a map, each mount point is able to have mount options. It is possible when doing multiple mounts to have a default that applies to all mount points and then have specific overrides:

/usr/local -ro,soft \	
/	node1:/export/local \
/bin	node2:/export/local/sun3 \
/share	node3:/usr/local/share \
/src           -rw,secure	node4:/home/jones/src

In this case, the default hierarchy option is a soft mount in read-only mode. At the same time, however, the source subdirectory (/src) is mounted in read-write mode with the Secure NFS option.



### *Multiple locations*

Multiple locations allow the Automounter to pick the first available server for a particular file system. Take the following map:

```

/usr/local \
/           node1:/export/local \
/bin       node2:/export/local/sun3 \
/share     node3:/usr/local/share \
/man       node4:/usr/man \
           node5:/usr/man \
           node6:/usr/man

```

The root, binary, and shared file systems below `/usr/local` are all specific file systems on nodes 1 through 3. The manual pages, however, are contained on three different servers, and the user doesn't particularly care which one is picked.

Multiple locations only make sense if the systems are identical. A read-only database is an easy example, but the system binaries might also be replicated. When the Automounter sees a reference to a portion of this file hierarchy, it will "ping" the null NFS procedure on each of the three servers. The first server to respond will be the one that is mounted.

This procedure is a rudimentary form of load-balancing. We can assume that the first server to respond is the one with the most available resources. If a server is busy, it will not respond as quickly. This is somewhat rudimentary, because we are only seeing which server is least busy at that exact time and are not taking into account average load or total capacity over time.

An alternative syntax for the multiple mount is as follows:

```

/usr/man      -ro,soft      node1,node2,node3:/usr/man

```

This syntax simply says that multiple locations are available for the same pathname.

Let's assume that node1 responded first and is therefore mounted. What happens when node1 becomes unavailable? By default, the Automounter waits for 5 minutes of inactivity and then unmounts the file. If the user does not refer to the inactive file system for 5 minutes, it is unmounted. A subsequent reference to `/usr/man` will result in another node being picked.

An alternative to waiting 5 minutes is to explicitly unmount the unavailable file system. When referenced again, the unavailable server will not respond to the RPC ping, and another node will be picked instead.

### *The Subdirectory Field*

One more feature of the map syntax can save time for the user (if not for the person writing the map). The real syntax of a map is actually:

```
key          (mount_option)      server:pathname(:subdirectory)
```

For example, the following is a valid map:

```
john          home_node:/home/home_node:john
mary          home_node:/home/home_node:mary
joe           home_node:/home/home_node:joe
```

When a user refers to `~john`, the Automounter will do two things:

- Mount `home_node:/home/home_node`
- Create a symbolic link `/home/john` which points to `/tmp_mnt/home/home_node/john`

If a user refers to `~mary`, the Automounter sees that the path is already mounted. All it has to do is add a symbolic link. Saving extra mounts means that the user sees a quicker response time.

### *Substitution*

We've referred several times to a convention of putting home directories into a `/home` directory to allow them to be easily changed. This can lead to very large maps in the case of large user populations. Two conventions save on writing large numbers of map entries.

First, the ampersand means that the key should be substituted into the map:

```
john          home_node:/home/home_node:&
mary          home_node:/home/home_node:&
joe           home_node:/home/home_node:&
```

What if all users had their own computers? In this case, the format of the map is really:

```
node          node:/home/node
```

At this point, the map can be brought down to one line by using the second convention, the wild card. The wild card matches any entry:

```
*            &:/home/&
```

A user can now change to any home directory on the network simply by knowing the user's name. Note that the wildcard is always a successful match, so it should be after any more specific entries in the map.

## Built-In Maps

A special built-in map in the Automounter is the hosts map. This map, referred to as “-hosts” in a map entry, uses the hosts database, which consists of the `/etc/hosts` file and the `hosts.byname` NIS map. This is a list of every hostname on the network. A typical use of the -hosts map is as follows:

```
mynode% automount /net -hosts
```

Note that this mapping could have been included in a master map or as a line in some other map instead of at the command line.

When a user changes to `/net/hostname`, the Automounter will mount all the accessible exported file systems of that host. Note that even if `/net/hostname` is mounted, that doesn't mean the entire `hostname` file system is seen: only the portions that were exported are shown. The exported file systems appear as a constructed hierarchical representation of the remote host. The Automounter will create directories as necessary to fill in gaps caused by unexported file systems.

The first thing that happens when a user changes to `/net/hostname` is that the Automounter will ping the null procedure of that host's mount service to see if it is alive. If so, the Automounter requests a list of all exported hierarchies from the server.

The Automounter will then sort the exported list according to the number of components in the pathname:

```
/usr/src
/export/home
/usr/src/sccs
/export/root/blah
```

This sorting ensures that mounts are done in the proper order that `/usr/src` is mounted before `/usr/src/sccs`. After all the systems are mounted, a symbolic link is returned to the top of the hierarchy.

Mounting all exportable file systems is not necessarily optimal for a user who only wants one or two files. The purpose of the -hosts database is to handle the majority of the cases where users do not know what they want. When something is repeatedly referenced, it is time to include that file system directly into a map.

The -hosts map and the use of a home directory convention means that users can now understand two things about the network:

- To see a node's files, refer to `/net/nodename`
- To see a user's files, refer to `/home/node/username`

Armed with these two pieces of information, a user is able to traverse the network without worrying about which node a particular user is on.

## When to Use the Automounter

In many environments, Automounter maps will be developed by the network administrator. The goal is to allow users on any workstation to see their home files without having to manually mount them consistently from any node. Tricks like the `-hosts` map and careful use of environmental variables and substitution allow a few lines in a master map to cover almost all situations.

Two characteristics of the Automounter make it much more useful than “raw” NFS. First, the ability to specify multiple mount locations means that the user is insulated from temporary unavailability of crucial file systems. If one copy of the file system is unavailable, the Automounter will automatically refer to the second. This also facilitates migration of file systems over time since the environment is now dynamically remounting file systems on demand.

Second, the Automounter is able to handle very large network environments in which the user does not know where everything is. By simply referring to a node name, the Automounter can pick out all available file systems and make that computer’s files available to the user. This means that a network administrator can simply refer a user to a particular node (“the software is on server5”) instead of having to go in and manually configure systems.

The Automounter is also quite useful for users. Since users do not need superuser privileges to make a map, they now have the ability to control which file systems they want mounted. Using manual mounts or the `/etc/fstab`, all users would have to have root privileges.

## For Further Reading

Brent Callaghan and Tom Lyon, “The Automounter,” *Proceedings of the Summer 1990 Usenix Conference*, Usenix Association, Berkeley, Calif., pp. 43-51, June 11-15, 1990.



## CHAPTER 12

# Distributed Applications



# Distributed Applications

## Overview

This is one of those grab-bag chapters which discusses a variety of seemingly unrelated topics. (Chapter 14, “Internet Applications,” is the other one.) Several chapters have been devoted to a single type of application: NFS, the Automounter, and messages.

Here are the distributed applications that didn’t merit their own chapter. This is not to say that the services discussed here are any less important than the others, only that this particular author has less to say about them.

We start with two infrastructure services. The license service is a way for vendors to dynamically allocate permission to use their software without tying the executable to any one particular computer. This makes sense on applications that may float from workstation to workstation but are rarely used full time by any one user.

The second infrastructure service is the lock manager. NFS, as we saw, is a stateless protocol, meaning that locks on data are not necessarily provided for within the NFS service. The lock manager extends NFS functionality by providing remote locking. The lock manager is a way for different applications to coordinate lock requests across the network. As with other Unix-like lock services, this is still an advisory service: it is up to the application to obtain the locks and then to obey them.

After the lock manager, we look at a series of commands which allow the user to access other computers on the network. The Remote Execution (REX) facility, which allows a suitably authorized user to borrow CPU cycles from the network, is explained. Also discussed are the  $r^*$  commands, part of the Berkeley heritage of SunOS. These commands allow the user to login, copy, and perform other services across the network.

Finally, we look at PC-NFS. The PC, by virtue of its total lack of architecture and planning, required some additional support utilities to integrate with NFS and Unix networks. For example, the reader may remember that

RPC bases authorization on the name of the user. That implies a login, a facility that most PCs don't offer. Several other features will be shown that require special treatment on the PC.

PC-NFS is more than just NFS. It is an implementation of ONC (NFS, RPC, XDR) and TCP/IP for the DOS operating system. The question of integrating the PC will be revisited in Chapter 16 when interoperability is examined.

## License Service

A license service is a set of software packages that grants a user the right to use a software program. This service may seem superfluous at first glance. Think, however, of an environment that is using a large, expensive product. Take the Ingres database system for example. You have ten people writing simulations, and you have five large Sun servers. One consequence of a system being large in terms of disk space, computational requirements, and other metrics is that it also tends to be expensive.

After a little financial analysis, you've decided you can afford two copies of the software. Which of the five servers do you put the software on? Wouldn't it be nice to be able to run Ingres on any of the five servers so that no more than two copies were active concurrently?

Concurrent, flexible usage of software products over a large network is the domain of the license service. An alternative to a concurrent license service is copy protection based on node locking: the serial number of the computer is somehow embedded into the software so it won't run on another computer. This is an inflexible scheme which becomes visibly painful if one or more of the nodes for which the software is licensed is taken off-line for extended periods of time.

There is an alternative to node locking: site licenses. Site licenses say that for an incredibly large amount of money, the vendor will let anybody in your organization use the software. If an organization is someplace in between General Motors and a one-computer site, the site license is not exactly cost-effective.

The SunNet License Service solves this type of problem. In order for the license server to work, however, software vendors have to write their software to take advantage of the license service instead of performing their authorization decision simplistically with site licensing or node locking.

The basic function of the service is to keep a library of licenses, each library containing an encoded token. The token, a piece of data encrypted by the software vendor, typically contains license units. The license units, in turn, grant the user the right to use the program anywhere on the network (subject to concurrent usage limits built into the license).



The concept of license units can be applied by the software vendor in any way it wants. More license units might be needed, for example, to run a piece of software on a Sun-4 than on a Sun-3. A new version might take more license units than an old version.

Notice that the service works over the network: software is really any abstract group of programs. A vendor could easily make different pieces of software, say one for each of the operating platforms. A price list is published, specifying how many license units are required for a piece of software on a particular platform.

When the software is executed, the first thing it does is use a license library located in the software executable. The license library finds a license server and requests some license units. The license units are given back to the software, which verifies that the license units are proper and then continues executing.

The SunNet License Service has a few other features besides the basic ability to grant, deny, or queue a license request. Expiration dates can be put on a license, allowing software to be available for demonstration for a limited period of time. A vendor could distribute many pieces of software, say on CD-ROM. Users could try the software for a while. After they have sampled the software, they call the vendor, obtain a license, and are free to use the software more rigorously.

### *Architecture*

There are four components to the license service. First, there is a license library which is linked with a vendor's application. Second, there is a license server, with the license database. Third, there is a license administration tool, used to install licenses and monitor usage. Fourth, there is the license production tool with which the software vendor produces the actual licenses.

The license library and the license server communicate with each other using RPC calls. Since XDR is used for the data, it does not matter if the license library is located on a different computer architecture than the server.

The calls available to the application program to communicate with a license server are shown in Figure 12-1. Notice that the calls are compatible with a *de facto* API developed by the Network Computing Forum, allowing a program written to work with the SunNet License Service to also work with another vendor's license server with minimal conversion effort.

Each server manages a distinct database of licenses. Note that a license doesn't float: it is always served by the same server. If the server is not operating, the license is not available.

This doesn't mean that an application is limited to a single server. An application can scout around the network looking for a server that has an

Call	Parameter	Description
ls_log_message		Put message in license system log file.
	my_job_id [IN]	The job ID of the calling program.
	length [IN]	Number of bytes of text to be written in this message.
	text [IN]	The actual message that is to be logged.
	status [OUT]	Either success, system error, or job_id not found.
um_get_record		Retrieve records from usage metering log. Any of the search fields may be wildcards. Only one vendor per search.
	my_job_id [IN]	The job_id of the calling program (thereby identifying the user and the vendor involved).
	job_id [IN]	The requested job_id.
	record_id	The requested database record ID.
	product_id	
	version	
	user	
	group	
	node	
	begin_date	
	end_date	
	maximum_length	Maximum number of data to be returned.
	length [OUT]	Number of bytes returned.
	data,	
	status	
	record_pointer [IN]/[OUT]	The next record to be searched for use in wildcard searches returning multiple values.
um_purge_records		Delete previously marked usage metering records.
	status [OUT]	Either success, system error, or job ID not found.
	record_pointer [IN]/[OUT]	The next record to be searched for use in wildcard searches returning multiple values.

## 12-1 SunNet License System Calls

available license for it. As soon as the license is granted, the application is ready to continue executing.

If a network is large enough, it may have multiple license servers. For example, one server could be for marketing, another for engineering. It is up to the network manager to decide how to distribute the licenses. For example, if the user has two licenses available, one might be put on each server. Alternatively, both licenses might be put on a single server.

How does an application know where to find the correct server? The license library in an application maintains a binding database that specifies which servers to consult when looking for a license (as opposed to randomly trying servers throughout the network). The binding database, a lo-

Call	Parameter	Description
<b>Concurrent Usage Calls</b>		
<b>lb_request</b>		Request a license for the specified application.
	my_job_id [IN]	Job_id of the calling program, thereby identifying the user and vendor involved.
	product_id	Product ID of the product to be executed.
	version	The version of the product to be executed.
	amount	An integer specifying the number of license units requested. The semantics of this number are up to the vendor.
	queue	
	check_period	The number of seconds the license system will keep a license available but unclaimed before the application is presumed dead.
	license_handle [OUT]	The handle for this transaction. Used to refer to this particular license during other calls.
	status [OUT]	Either license granted, license not available (no queueing), no license available (queued), or system error.
<b>lb_wait_remove</b>		Remove an entry in a license wait queue.
	license_handle [IN]	License handle of the request to be removed.
	status [OUT]	Either success, request not found, or system error.
<b>lb_release</b>		Release a license.
	license_handle [IN]	License handle of the request to be removed.
	status [OUT]	Either success or system error.
Source: Recommendations for Network Software Licensing Practices, Version 1.0, The Software Licensing Working Group, Network Computing Forum, June 1, 1989.		
<b>Additional SunNet Server Calls</b>		
<b>nl_init</b>		Initiate a licensing session.
<b>nl_term</b>		Terminate a licensing session.
<b>nl_check_que</b>		Renew position on queue(s).
<b>nl_confirm</b>		Renew license.
<b>nl_binding</b>		Cause binding file to be built.
<b>nl_bind_by_vendor</b>		List of servers with license for a vendor and product.

## 12-1 (Cont.) SunNet License System Calls

cal or NFS-accessible file, contains a product to server name mapping. Server names can also be defined in an NIS or NIS+ table. In NIS+, the user can add custom information to the tables.

The requirement for a license to be served from a single host is a security feature. The host name of the server is built into the license before the vendor ships it to the user. When an application checks out a license, it



Vendor
Product
Version
Serial number of license
Host name
Domain name
Host ID
Number of units
Expiration date: date license not valid
Maintenance date: when maintenance contract expires
Which previous versions supported?
Vendor private data

**12-2**

## Contents of a License

looks inside the license at the host name and then checks to see if that matches the name of the host that actually furnished the license. Several other checks, such as the proper domain and the availability of a valid number of units, are also conducted.

Why do this? It prevents the situation of a masquerade by moving a license to several different license servers. The protection is not foolproof: two servers could have the same host and domain names in separate universes. Of course, while the license server would work properly, an awful lot of other applications would have problems with duplicate host and domain names.

*The License*

A license is a data structure inside of the license server that identifies each license by the vendor, product, version, and serial number. Associated with the license are a number of available units (see Fig. 12-2).

As a general rule, the unit represents a user, but there is no reason why it can't be weighted by time of day, type of computer, or day of the week. This policy is up to the vendor of the software. All the license server does is manage units and decide if they are available.

Also in the license are client entries, each one representing a user of the software. A client entry includes the number of units in use by this user, the name of the user, and the time of the next license renewal.



## License Service Operation

The first thing that must happen to use the license service is that the software vendor needs to alter the application to use the license library. The software is then delivered to the customer. The second thing that has to happen is for the vendor to produce a license: a file. The file is included on the software tape, sent by electronic mail, sent on a floppy disk, or downloaded via modem. The network manager takes the license and installs it using the License Administration Tool. The user then tries to run the application.

When the user tries to run the application, the initialization code will make several calls to the license server. It will request a license from one or more servers. If a license is not available and queueing is appropriate, the request is queued on one or more servers. Note that the decision to queue is up to the application. The application might make this decision unilaterally or might consult the user.

Once a license is available, the application runs normally. Periodically, the license is renewed. Finally, when the user is all done, the license is released. Of course, if the application is not well behaved, it might forget to release the license. This might also happen if the node on which the licensed application was running on crashed before yielding the license. In both cases, the server would wait until the next renewal limit is reached and then make the license available to somebody else.

Releasing a license is up to the application, but the well-behaved application will release the license in three circumstances:

- When an application is idle for a period of time
- When an application converts to an icon
- When the administrator requests a return

Releasing a license raises some interesting policy issues. For example, if the user clicks on the application, it will not instantly return, or will return with a message that no licenses are available.

Even trickier is when an application has been idle for a period of time. Let's say the user goes to lunch, having forgotten to save the current work. When the user returns, a license is not available. Should the user be allowed to save the work? Again, it's up to the application. These are issues that application developers must address (and hopefully minimize) when adding license services into their applications.

## *License Library Operation*

The license library is responsible for handling the interaction with license servers, shielding the application from these details. The library queries the servers listed in the binding database to find a license. The library accepts

the first license from the first server with license units available. It takes the license and decodes it (a part of the license is stored in encoded format in the server database).

Decoding licenses is based on a License Access Module (LAM). There is a default encryption method, but there is no reason why the application can't use its own encryption method. When the license is decoded, the library validates information in that license to make sure it hasn't been altered and that the server has the right to serve this token. The license library also checks to see if any of the checked-out units in the token have expired: if so, the license library contacts the server, frees up those units, and returns them to the pool. The license library also takes vendor private data out of a license and returns it to the application.

### *License Administration*

The license server process is started by the internet daemon (inetd). It uses the NetISAM RPC service as a database manager. NetISAM is a general networked file service available on Sun platforms. In the case of the License Service, NetISAM is bundled in; it is not a general runtime license.

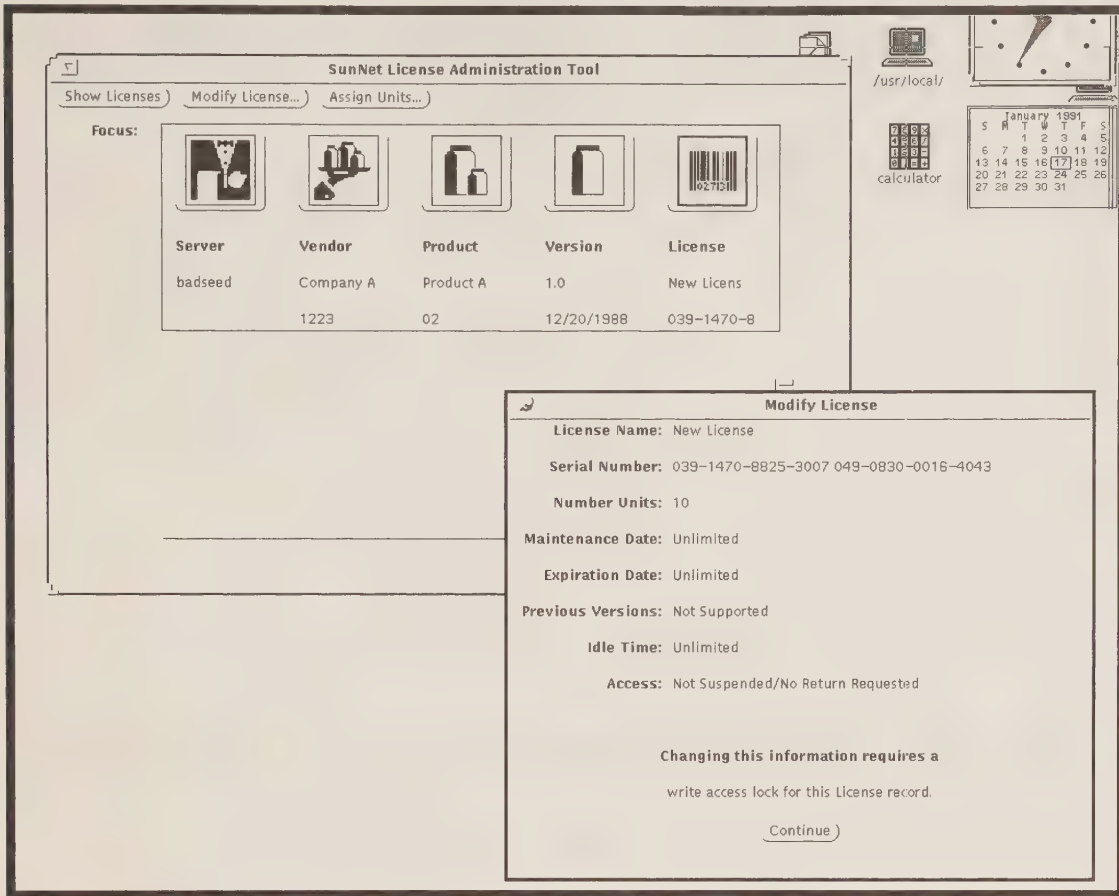
The server itself is not licensed. It is bundled with the operating system. The cost of license servers is born by the vendor who licenses the library at a one-time charge. The software vendor can then redistribute the service as needed.

The License Administration Tool (LAT) allows the network manager to administer licenses and install them. The tool uses the SunView or OpenLook interfaces running on a bit-mapped display on a Sun workstation. Figure 12-3 shows an example of this tool. In this figure, the user is modifying a license on a server.

The LAT can be used to manage any of the license servers on the network. Presumably in a large network, this tool would be located in two places. The purchasing department (or the people who receive software) would use the tool to install licenses as received. The network manager would also use the tool to monitor license usage and thus decide when new licenses are needed (see Fig. 12-4). The network manager can also use the LAT for constructing access lists.

The access control list allocates license units based on users or groups of users. For example, if an organization has 21 units, 10 might be allocated for engineering, 10 for marketing, and 1 for the boss.

The license production tool is software that runs at the software vendor's facility. The tool is based on the curses library, a character-handling library that comes from early Unix releases, meaning that it will run on almost any terminal. The license production tool uses the application's License Access Module to encode the token and place it into a file (see Fig. 12-5).



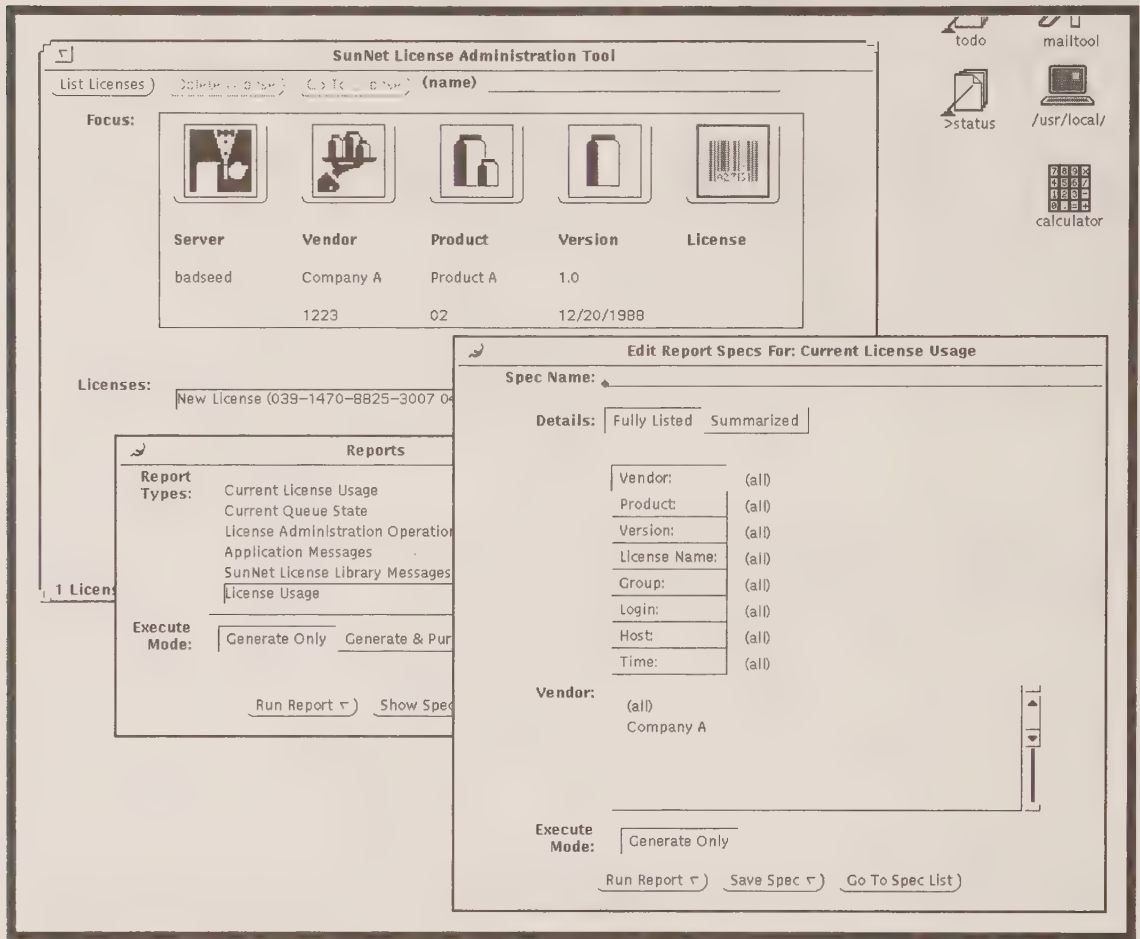
### 12-3 Modifying a License

The file is then moved over to the user's site where somebody installs it on a particular server. The information that the software vendor needs to provide to the license tool would be the license server name, the domain name, and the host ID.

An example of the use of the SunNet License Service is Sun's own CG3270 software. This software allows a Sun workstation to open up a color graphics 3270 emulator window to an IBM mainframe. When the software starts up, it gets its license from the license server, thus allowing any user (subject to access control and units available) access to the software.

### Network Lock Manager Protocol

Locking is the ability to prevent another program from performing an action on some object. The object is usually a file or a record inside that file,



## 12-4 Requesting a License Usage Report

but the concept can be extended to include other types of objects. Locks regulate the level of concurrency allowed.

A typical lock is a write lock, also known as an exclusive lock. Obtaining that lock means that nobody else can access the object until the user is through with it. Other locks are read locks, also known as shared locks. When writing data, it is not good for others to be able to read the information because the data may be inconsistent: the update may be only halfway applied. With read locks, it is acceptable for others to share in the reading of the data, but a write lock request would be denied.

Notice that locking helps develop multistatement transactions. An atomic update is when only a single piece of data is changed. Each update is totally independent from any other.



```

$ shelltool - /usr/local/bin/ksh
SunNet License Production Tool
Make License from (F)orm Menu
(H)elp; (R)eset; (C)reate this license; go to (M)ain menu

Command: 

Vendor Name: Company A
Product Name: Product A
Version Number: 1.0
Expiration Date: any
Maintenance Date: any
Previous Version: UNSUPPORTED
Total Units: 10
Server Hostname: badseed
Server Domain: ebb.eng.sun.com
Server Hostid: 5300a42b
Customer Name: sample
Comments: this is a license for sample program

<to pick an option, please type the letter shown within the parentheses>

```

## 12-5 The SunNet License Production Tool

For an environment characterized by atomic updates, a lock manager is not really necessary. All actions are considered idempotent: they will affect the data even if the updates are out of order or repeated.

An example of an atomic update is “change the value of bytes 10 to 20 to contain the value 10,000.” No matter how many times or when this update is applied, the resulting value will always be 10,000.

Now, think about a banking application. Let’s say the prior application represented a deposit of \$1000. Obviously, the current balance is going to be the prior balance plus the deposit. The program gets the prior balance of \$9000 (a read operation), adds the deposit amount to it, and then writes the current balance to reflect a value of \$10,000.

What if in the time between reading and writing, somebody else wants to make a deposit to the same account? Let’s make it simple and say it’s another \$1000 deposit. We now have two programs that each read a balance of \$9000.

Each application adds \$1000 to \$9000 and submits a request to change the value to \$10,000. The file system takes each of the two write requests and applies them, yielding a final value of \$10,000. Not a situation the customer is pleased with.

This example shows that two related operations helped form the final transaction. Doing one operation without the other yields an inaccurate result. In our example, we would want an exclusive lock on the data to prevent the second program from performing the read on data that was about to be changed.

In Unix operating systems, protecting the integrity of the data is normally left to the application. This means that the file system will grant any request to read or write data. It is up to the application to coordinate its request with other users (and possibly other applications) to make sure that inconsistent updates do not occur.

In many other operating systems, locking is built in at a lower level. In DEC's VMS, for example, the file system coordinates lock requests. When a user opens a file, that request is submitted to the Record Management Services (RMS) along with the desired lock status. RMS will make sure that the request is consistent with other currently executing requests and will then grant the lock request.

In both cases, the lock manager is a separate program. The lock manager doesn't manage files: it manages concurrent requests for some abstract object (which may contain other objects). In a file example, this is simply files and either records or blocks inside the file. In a database management system, locks might be tables, rows, columns, or instances of data.

The important point here is that some program has to enforce locking. The program interacts with the lock manager (which interacts with other programs) to see if some object is currently accessible. Once the lock is granted, somebody has to see that it is honored.

In the VMS example, RMS acts like a central cop, making sure none of its users can perform an action for which it has not obtained an appropriate lock. In the Unix example, a program obtains a lock and then honors that lock—it will not try to write data in a record for which it does not have a write lock.

### *The ONC Lock Manager*

The lock manager is one of several RPC services available in a distributed environment (NFS, REX, and several others are also available). The manager provides advisory locking of files: the lock manager acts as a coordinator for programs that choose to use the lock manager. If programs bypass the lock manager, of course, coordination is left up to the applications.

The service takes the lock calls that are part of the System V Interface Definition (SVID) and makes them available over a network. A program can use NFS to access remote files in the same way as local files. The lock manager allows the program to access remote lock services in the same fashion as local lock services.

Do Nothing	
Test Lock	Accepts a description of the lock desired and returns whether the lock would block or not.
Establish Lock	Accepts information of the entity to lock, if the request should be queued, if the lock is exclusive, and if the lock is a reclaim request.
Cancel Lock	
Unlock File	
Grant a Lock	Sent by the server to grant a request.
Share a File	DOS enhancement to set the access mode for a file as read-only, write-only, or read/write. These requests are not queued: if denied, the client must retry later.
Unshare a File	
Unmonitored Lock	DOS enhancement to set up an unmonitored lock. The server will not attempt to notify the PC when it has crashed and rebooted.
Free All	PC sends this call when it reboots to cancel any locks that have been outstanding.

## 12-6 Lock Manager Calls

The SVID lock definition provides for file and record locking. In addition, the ONC lock manager was enhanced to support the MS-DOS file sharing capabilities, allowing PC applications to also use the lock manager.

The network service relies on the client to keep the current state. If a client obtains a lock, it is the client that must remember that fact. If a server crashes and then reboots, the clients will have to reclaim their locks from the server.

Finding out when either a client or a server crashes is vital to network-based locks or any other stateful protocol. Remember that in NFS the protocol is stateless: if either client or server crashes, no recovery actions are needed.

To support stateful applications, a separate service called the status monitor was created, which is responsible for deciding if a remote computer is still functioning. The lock manager worries about coordinating lock requests, and the status monitor keeps track of the current state of the remote clients.

The basic operation is quite simple. A client submits a lock request to the remote server (see Fig. 12-6). If the request is granted, the client assumes it has the ability to perform an appropriate operation on the remote object without fear of incompatible operations occurring simultaneously. Normally, the lock will then be released by the application, making the resource available to other users. One hopes that the lock release will be



quick, but applications have been known to lock files while their users are off on extended lunches.

If there is a problem with a node, the status monitor informs the client or the server. If the server crashes, the client has the choice of waiting: in a database example, the client might have been half way through a multi-statement transaction, and it needs to get to the server to either reverse or complete the transaction.

When the server reboots, it starts with a clean slate where all locks are released. The server will not accept any requests for new locks for a grace period. During that time, the client that has locks on the recovered server is notified that the server has rebooted and can reclaim all prior locks. After the grace period is over, the server resumes normal operation and continues to coordinate new lock requests.

While there is a reclaim time for server crashes, when a client crashes, the server will release all locks that are held by that client. Since the resources being locked are on the server, it makes sense to avoid keeping them unavailable for an indefinite period of time while the client recovers since the applications would have crashed at the same time as the client they were running on and would have to restart anyway.

The basic call is to obtain a lock on a file or a record inside that file. A flag on the remote procedure call allows the client to reclaim a prior lock. Obviously, this flag only makes sense during the server grace period and is ignored during normal operations.

Once the operation is completed, the user submits the unlock request. It is also possible to test whether locks would be granted. Of course, between the time the test message goes back and a decision is made to get the lock, somebody else might have beat the program to it.

Identifying the lock requested is handled the same way as it was in NFS. The client gets an opaque piece of data, known as a cookie or file handle. The cookie means something to the server: for a Unix server, for example, it includes information such as the I-node that corresponds to a particular file.

A lock request also includes a length and an offset. Notice that in Unix there are no records provided by the file system. Whether records are fixed or variable length (text followed by a carriage return for example) is up to the application. Since the file system only provides a block of data (defined by the length and the offset), it makes sense for the lock manager to provide the same service.

A lock, once granted, is identified by two pieces of data: the file handle identifies the object, and the owner handle identifies the owner of the data. While the file handle is opaque to the client, the owner handle is opaque to the server.

Basic SVID calls work well for a Unix environment, but other operating systems have different locking semantics. In particular, MS-DOS added the



concept of file sharing (a radical new concept for this brain-dead operating system), which is combined with the file locking capabilities. The file sharing says what other users can do; the file locking says what you can do. The file sharing mode thus affects future concurrent lock requests.

A typical example of combining the two is when a file is opened (locked) for reading and writing, but other users are allowed to read the data. We don't want other people writing, but we don't mind if they read the data (even if it is inconsistent).

## REX

REX is the remote execution service, a service built on top of the RPC mechanism. It is somewhat similar to the rsh command in 4.3 BSD Unix in that a command is run on a remote computer. REX has a few significant advantages over rsh.

The basic idea behind REX is that a command is sent over to some other computer and the command is executed. The results show up back on the client system. This is a way of taking advantage of CPU resources on other systems. A more sophisticated use of REX allows the user to work across multiple machine architectures.

To understand rsh and REX, we need to briefly look at how programs execute on a Unix system. The user interface in Unix is a shell: a command processing environment. The shell has a series of environmental variables, things like the current working directory or the user's preference for how the time should be displayed.

When a shell is started, it looks in a local startup file for a startup script: a set of commands that (among other things) set up the environmental variables. It is not unusual to have multiple shells active concurrently: several background or stopped jobs and the current foreground job. In the case of a windowing environment, each window could be a separate shell.

The idea of the rsh command is to allow a shell to be executed on a remote computer. The user must have an account on the remote computer or specify the username-password combination on the command line.

In addition to access control information, the user includes the name of the remote host and the command to be run. The rsh command takes the information and places a TCP connection across the network. In effect, the user will have logged onto the remote machine.

This is much more convenient than just logging in: that takes some steps by the user. The rsh command logs in just long enough to run the command desired and move the answers back to the client computer.

What is significant here, however, is that a shell is started on the remote machine. This takes a bit of time. Equally important, however, is that the

REXPROC_START	Starts a REX session.
REXPROC_WAIT	Terminates a REX session—indicates that the client will make no further requests during the current session. Allows the execution to complete and returns the command's result status.
The following three calls are used in interactive mode:	
REXPROC_MODES	Initializes virtual terminal modes (e.g., specifies which erase and interrupt characters to use).
REXPROC_WINCH	Signals changes in the size of the window on the client.
REXPROC_SIGNAL	Passes the interrupt signal to the server.

## 12-7 REX RPC Calls

local environmental variables and local files are not available unless explicitly moved to the remote machine.

REX can be thought of as an extension of the local shell. Instead of starting up a remote shell, REX preserves the local environmental variables and brings those over to the remote machine, using RPC calls. No server needs to be activated (although the REX daemon must be already active), and no shell needs to be started up.

REX does more than just bring over environmental variables. It uses NFS to mount the user's current working directory on the remote machine. Local files are all available. Instead of explicitly being on another machine as in rsh, REX is more like borrowing the remote CPU for a few minutes in order to supplement the current machine.

Once the working directory is mounted on the other machine, TCP-based connections are set up and are matched to the three basic channels used in Unix: standard in, standard out, and standard error. Mounting the working directory and connecting the basic channels is important because it allows REX-based commands to be used in conjunction with local commands. For example, the results from a local operation can be piped into the remote operation being executed via REX. Errors can be redirected into special files, and the output can be piped into yet another command.

The fact that REX is an RPC-based service is significant (Fig. 12-7 shows the RPC calls that underlie the service). The underlying RPC credentials are used to verify a user. The remote shell command, by contrast, uses a very simple UID-based scheme, essentially only looking at the name of the client host.

REX uses the TCP transport service underneath RPC. TCP means that very large amounts of argument data can be passed between client and server. UDP, by contrast, limits size of data to 8 kbytes. Why does REX

need all this? Remember that it passes the entire shell environment through to the server.

REX is made up of five RPC procedures. Each procedure is called by the on program and acted on by the server daemon (REXd).

The basic call is REXPROC\_START. This passes the shell environment variables, the current working directory, and the command to be used. It also includes the three ports on the client that are used for input, output, and error streams: the server will link those three ports up to the command being executed. For REX to work on a system, it must have been started in the internet daemon configuration file (inetd.conf).

A REX user must be known to the server, which will use RPC Unix authentication to establish the identity under which the command is to be executed. We assume that both parties share the same NIS domain (or that there are local password entries for the user on both machines).

There is also a Secure RPC version of the REX command. Secure RPC REX takes care of a user being authoritatively identified to the remote server. In addition, Secure NFS should be used to ensure that the files are securely mounted.

There are two pieces to REX: the client and the server. On the client there is the "on" command, which can be used as a batch-style or interactive command. In batch-style, the command is executed and the results returned, as in listing the names of files:

```
myhost% on bighost ls
```

The command says to run the ls command on the host bighost. What is interesting is that when the ls command runs, it will list the directory contents of the current working directory from where the on command was launched. This is because the ls command with no parameters specifies the current working directory. NFS is used to remotely mount the current working directory on bighost.

The interactive version of the on command is used for running programs that allow a user dialogue. For example, vi could be used on a remote host to look at some file:

```
myhost% on bighost vi /usr/src/bigfile
```

This command will run the vi process on host bighost until the user leaves vi. Here the file is relative to the remote host: this is not the /usr/src on the local file system.

This last statement may seem at first to contradict the first statement about the on command. When the on command is run on a remote host, only the current working directory on the local host is mounted using NFS. All other file references are to the remote file system. The reason for this convention is that it preserves machine independence. Take the /bin direc-



tory, for example. This directory contains the executables for common Unix commands. However, each executable contains instructions for the machine architecture on which it is based. The Sun-3 uses a Motorola 68000 series chip, for example, whereas the SPARCstation has the SPARC architecture.

The default is therefore to use the files on the remote machine, since any programs will thus match the architecture of the remote CPU. However, there is one exception. The current working directory usually contains data: source code to be compiled, documents to be edited, or mail to be read. It thus makes sense to use the current working directory along with the remote file systems.

Needless to say, this can all be changed with creative NFS mounts. Careful mounting and cross-mounting can make needed files available. The Automounter, combined with local naming conventions, can help make needed files available when referenced.

Let's take a look at REX versus rsh once again. Remember that one reason that REX executes faster than rsh is that it ships over environmental variables instead of having to set them up on the remote host.

Two points should be made here. First, the performance difference between the two applications decreases as the command takes longer to run: there is no speed in execution, just a difference in set-up time. Second, it is assumed that shipping the data over the network is faster than initialization: if there is a complicated local environment and an uncomplicated remote environment, the shipping time may exceed the setup time.

A way to illustrate the differences between the two commands is using the print environment command. This command shows the value of an environmental variable. We can combine this command with the /bin/time command which times command execution:

```
myhost% /bin/time rsh compuhost printenv HOST
compuhost
          9.2 real      0.1 user      0.2 sys
myhost% /bin/time on compuhost printenv HOST
myhost
          2.6 real      0.2 user      0.3 sys
```

The first command specified that the system should time the rsh command and then have the rsh command run the printenv HOST command. Thus, two types of output can be seen. First, the variable host has the value "compuhost" on the remote computer. Second, the time it took to run was 9.2 real seconds, of which 0.1 CPU seconds were used for the user and 0.2 CPU seconds were run on the system.

The second command specified that the system should time the on command and then have it run the printenv command. The result here is my-



host, since the local environmental variables are shipped over with REX. The amount of real time is significantly less, although notice that more CPU time is used.

### *Administering REX*

For REX to work properly, a few things should be kept in mind. First, REX will try to mount the current working directory. This means that the working directory (or some directory higher in the file system) must have been previously exported.

When REX runs, all path and file names are relative to the server machine except for the current working directory. A bit of care must thus be exercised. For example, if a C compiler is resident in the local directory, that compiler will be run when the following command is issued:

```
myhost% on compuhost cc file.source
```

If the machine compuhost is a different kind of CPU, this command will yield interesting results.

A typical example of REX is used in software development. Let's say you have a VAX, a Sun-4, a Sun-3, and a SPARCstation and you are developing a program which will ultimately be deployed on all machines.

It makes sense to develop the code on the personal SPARCstation and make it run before starting to worry about cross-architectural issues of running on multiple platforms. Once the code works properly on the SPARCstation, REX allows that code to be compiled on each of the remote platforms, producing executables appropriate to each architecture.

Another common use of REX is to take advantage of idle resources on the network. For example, your boss believes he needs the latest Sun-4 workstation, leaving you, the system manager with a Sun 2. Like most managers, this gentleman spends most of the time on the phone or in meetings. The solution is fairly easy. Rather than switching workstations and hoping the manager doesn't notice, the remote resources can be used with REX. Need to sort very large amounts of data? Use the `on` command (and put the command into the background so that you can continue working).

If the manager comes back, he may notice that his machine is running a bit slow. This is easily explained away with vague remarks about the weather and how climatic conditions are probably affecting the speed of his CPU.

### *The R\* Commands*

In addition to the remote shell (`rsh`) command, there are several other commands in Berkeley Unix including the `rlogin` (remote login) and `rcp` (remote copy). All three of these commands are meant to be convenient ways of

executing services and accessing data in the confines of the local network. It is important to recognize that the addition of a distributed file system like NFS, particularly when coupled with support services like Automounter and NIS, severely diminish the need for these ancient Berkeley commands.

Occasionally, the `r*` commands are used in a broader environment, but security considerations make these unlikely candidates for outside access on any network that has any valuable data.

For many commands, a user uses a username and a password to access services on a host. This is the case for Internet applications such as FTP and Telnet. Chapter 11 shows that sending a password is not always good, but it is certainly a line of defense. This line of defense gets to be a bit cumbersome in a local network. If a user has half a dozen accounts on half a dozen computers, it makes sense to start using all these computers as a single working environment.

The Berkeley `r*` commands allow this easy access. To give access to remote hosts that don't need a password to access local services, the name of the host (or of a group) is placed into the `/etc/hosts.equiv` file. This file establishes a machine as a trusted host. For example, the file may contain the following lines:

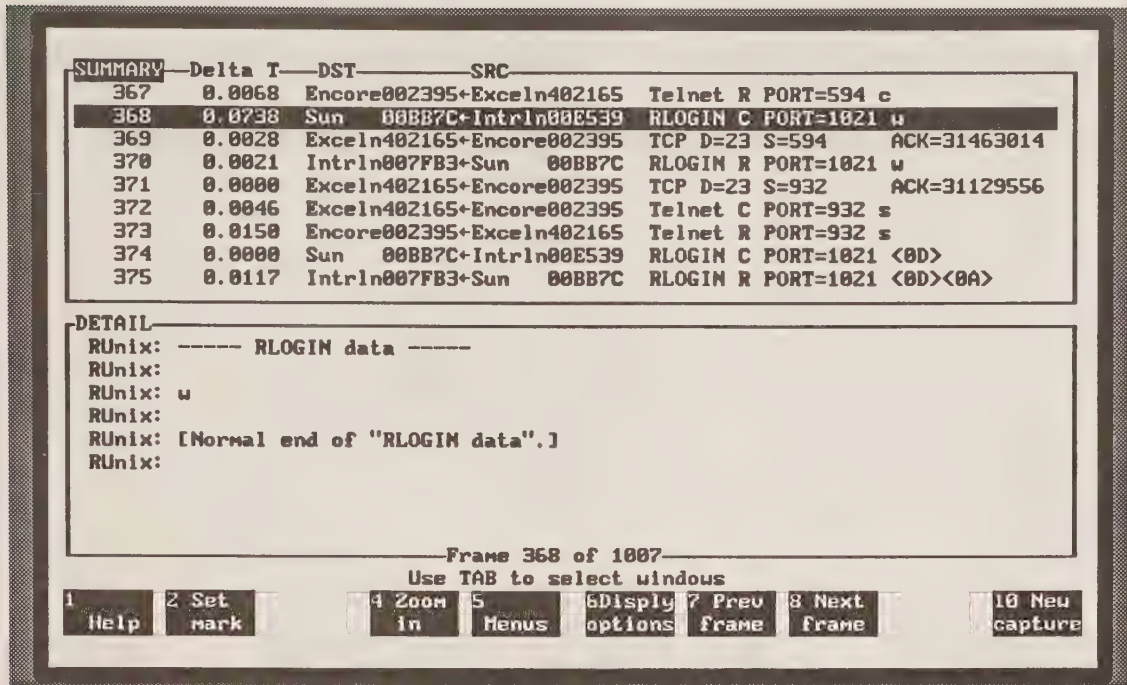
```
host1
host2    user_a
+@group1
-@group2
```

The first line allows anybody with an account on `host1` to access this system. The second entry indicates that only `user_a` on `host2` is trusted: everyone else must supply a password. The last two lines indicate that we'll trust everybody in `group1` and will not trust anybody in `group2`. The groups are defined in the `netgroup` database.

The second place that security is defined for the `r*` commands is in a user's home directory in the `.rhosts` file. This file is in the same format as the `hosts.equiv` file. If an entry in `hosts.equiv` denies a user access but the user's `.rhosts` grants it, the user has access.

The three commands that are most often used are `rsh` (remote shell), `rlogin` (remote login), and `rcp` (remote copy). There are several other commands (notably `lpr` for printing jobs) also used, but they are discussed in Chapter 14. Figure 12-8 shows an example of the `rlogin` command. Notice that the `rlogin` traffic is mixed with Telnet traffic on the same subnetwork.

There is one other command which comes from the same heritage as the `r*` commands but is not quite as popular with network managers: the `rwho` command. The purpose of this command is to know who is on the network (the `finger` command discussed in Chapter 14 provides a similar function).



12-8 Rlogin Traffic

The `rwwho` command received its notorious reputation not because of what it was trying to do but how it did it. Figures 12-9 and 12-10 show the `rwwho` command. As can be seen in Figure 12-9, it provides information on who is logged in (and how much idle time has accumulated for their session to distinguish active users from those who have gone to lunch).

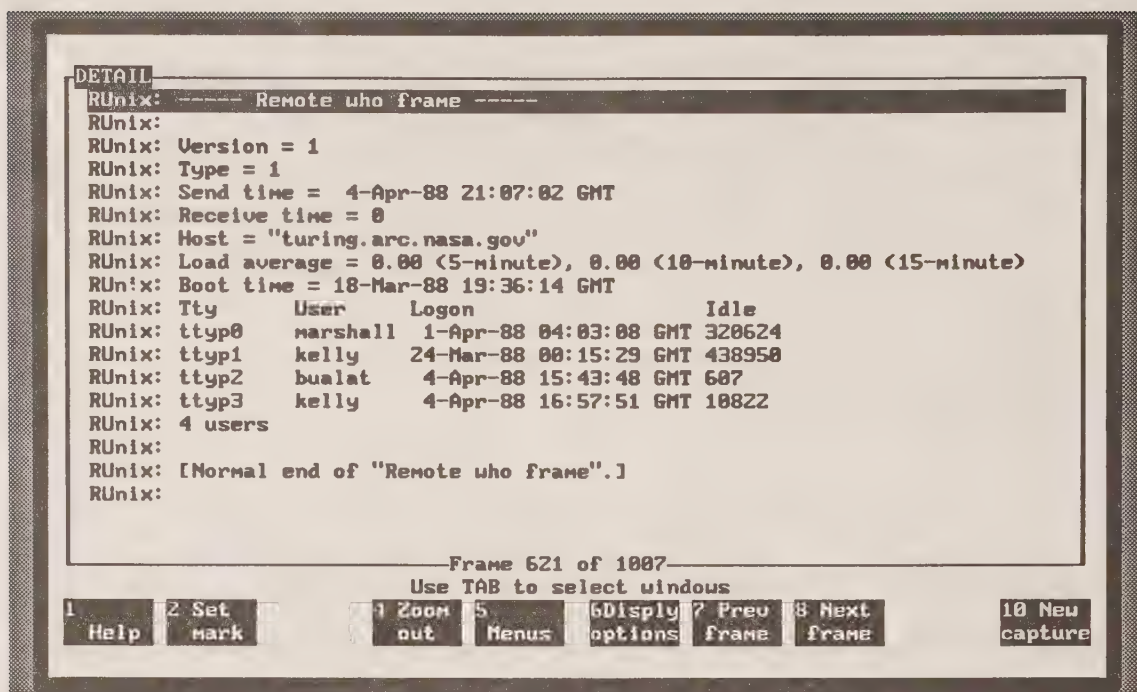
Figure 12-10 shows us how this is accomplished. Notice that every host is broadcasting this information to the network. This is known as an egotistic protocol design: the protocol assumes that others on the network actually care that we are logged in. On a network with many workstations, the amount of `rwwho` traffic can really add up.

Many network administrators use `finger` instead of `rwwho`. `Finger` waits until a user requests some information, then goes to the host in question and finds out the answer.

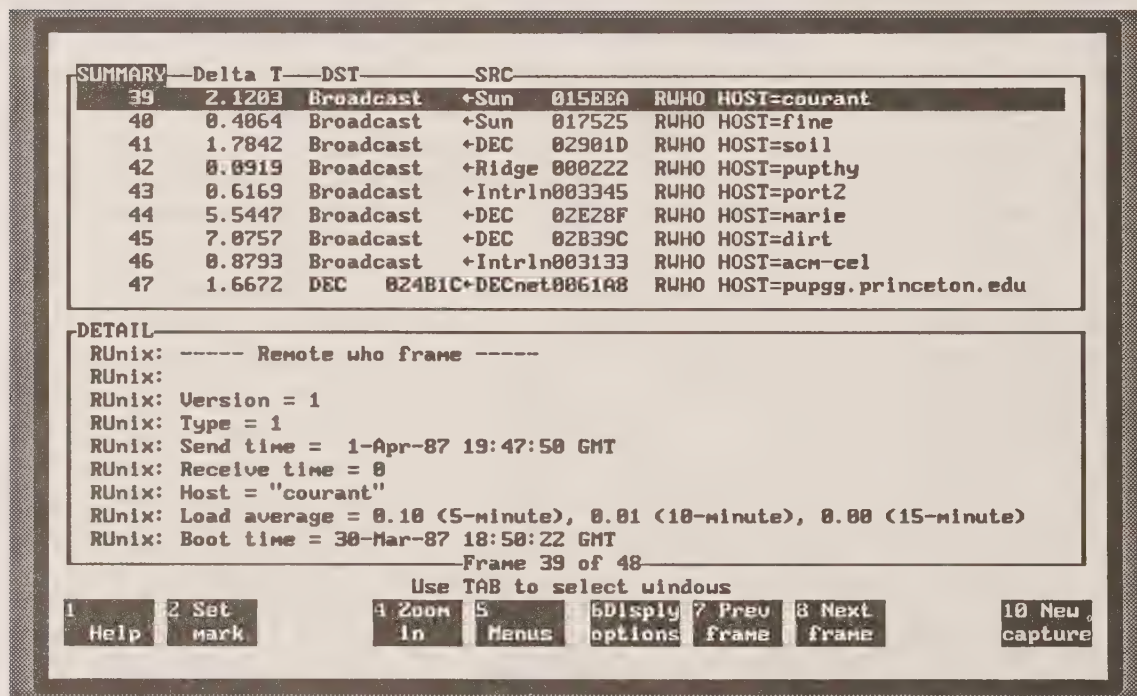
## The PC

PC-NFS can be thought of in two senses. In a narrow sense, it is the ability for a PC to use an NFS server. The package itself, however, includes several facilities in addition to just NFS. A print server is included and there is also a network-based backup utility and a mail system. In addition there is a PC-NFS Programmer's Toolkit (PTK) which provides programmers with a





12-9 The Notorious rwho Protocol



12-10 Gratuitous rwho Broadcasts



general RPC interface to the network. An additional package called Lifeline is available which allows access to Unix mail off-line from a PC-NFS client.

The user part of PC-NFS has been integrated into the Microsoft Windows 3.0 environment. This means that a user can add and delete network-based disk drives and icons from within the Windows control panel.

### *PC/NFS*

Work on a PC version of NFS began in the summer of 1985. At the time, all the ports of NFS were based on some derivative of the Berkeley 4.2 BSD Unix. As many readers may remember, the services provided by DOS in 1985 were somewhat, shall we say, rudimentary.

Original estimates by PC/NFS engineers were that a full-blown version of TCP/IP for the PC would take 150 kbytes of memory. Total available memory for DOS is 640 kbytes, of which 30 kbytes can easily be used up by the config.sys file. At the time, terminate and stay resident utilities (TSRs) were popular, taking off at least another 30 kbytes.

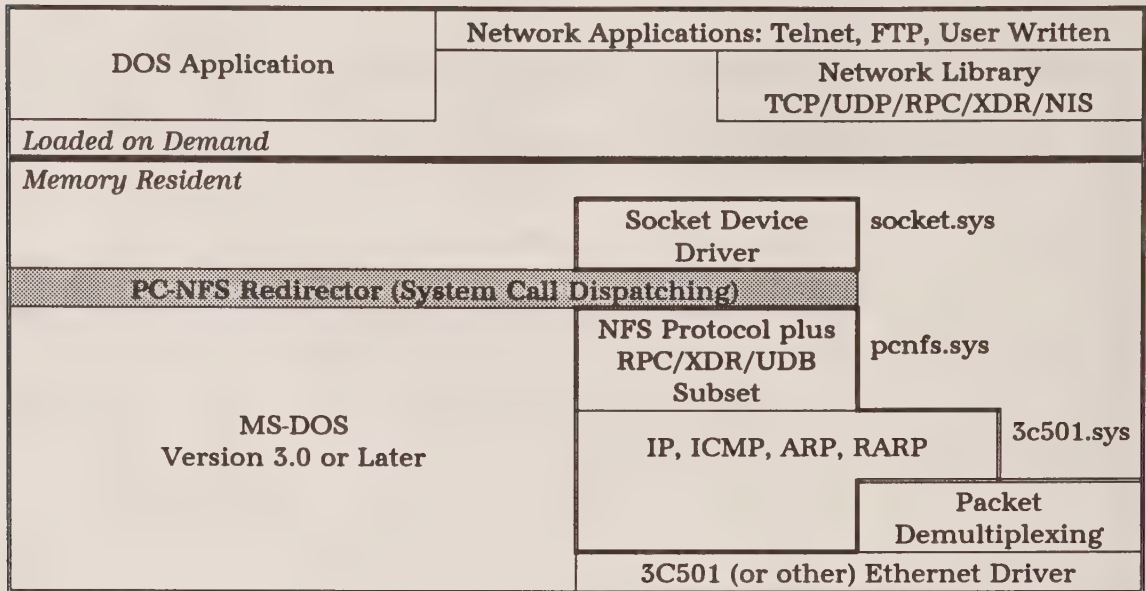
This left 430 kbytes available. Take out 40 kbytes for the DOS command process (command.com) and you have 390 kbytes. You still need to run a program like WordStar or Lotus, not to mention the need for some data. Conventional wisdom for PCs is that you needed to leave at least 500 kbytes of memory available for programs.

An alternative, of course, was to depend on smart adapters: Ethernet boards that had TCP/IP built right onto the card. At the time, most Ethernet controllers were far more primitive, and basing PC-NFS on the availability of smart cards would drastically limit the potential audience.

Memory was just one of many constraints for implementing NFS on a PC. The PC has no multitasking capabilities, making it tough in a network architecture that assumes a node is listening on an Ethernet port. But, when the packet comes into the Ethernet port, it has to be processed by the computer, or the buffer space on the controller gets saturated and packets start getting lost.

In order to do NFS, then, you had to live within the memory constraints. In addition, in order to do multitasking, the software had to intercept clock requests so that it could poll the Ethernet card to find out if any packets had arrived. Many TSRs on the market also intercepted the clock interrupt, and some of them would not be terribly well behaved. Instead of passing the interrupt on to the next TSR in line so it could do its work, the first TSR would hand the interrupt back to the main program.

One more implementation problem is the difference in the user bases. For VAX VMS and Sun Unix, most users have a raft of information that they consult regularly: on-line manual pages, tutorials, and local gurus. In other words, the Unix users are expected to take some time to learn how things work and to configure the software to their special environments.



12-11 PC-NFS Architecture

The PC world, however, has a very different user base. Here, things are expected to install themselves and operate in a fairly easy fashion. Very little customization and configuration can be expected from the individual PC user. Take a simple example of this. The user wants to use Lotus 1-2-3 to operate on a spreadsheet stored across the network. What if the server becomes unavailable? You can bet that if Lotus got a "Server unavailable" message instead of a "Drive Not Ready" error, it would crash.

### PC-NFS Architecture

The basic PC-NFS architecture includes support for an NFS client, the Telnet and FTP services, the Berkeley remote shell and remote copy services, remote printing, and authentication. Since the PC is the slowest machine on the network, things like modular design are often sacrificed in the interest of squeezing as much performance out of as little memory as possible.

The system consists of three different DOS device drivers (see Fig. 12-11):

- The pcnfs.sys driver
- An Ethernet controller driver
- The socket.sys driver

The pcnfs.sys driver handles the bulk of the PC-NFS work. It includes the IP and UDP layers of TCP/IP, including ICMP, ARP, and RARP. It also in-

cludes a minimal subset of RPC and XDR, just enough to handle the NFS and lock manager calls. This is basically the core of the network through presentation layers. Additional functionality, such as full RPC services, is handled by libraries that are linked in with application programs.

The second driver is the Ethernet controller driver, originally `3c501.sys` for the model number of the 3Com Ethernet card supported in the original implementation. The IP layer, implemented on the `pcnfs.sys` driver, would call the send routine of the `3c501.sys` driver.

When a packet is received on the Ethernet card, it goes to the `3c501.sys` driver. The driver reads the header and passes the packet to an input demultiplexing routine. That demultiplexing, usually an IP layer function, is bundled in with the Ethernet layer as an efficiency measure.

The demultiplexing code figures out which client gets the packet. Each client is identified by a data structure called a selector. The data structure may also point to a buffer to hold the incoming data. If a selector is available, the driver will copy the data into the buffer and call the client. The client is able to then schedule packet processing (and allocate a new buffer). If a buffer is not available, however, the driver is told to drop the packet.

There are two broad classes of selectors: those within the `pcnfs.sys` library (e.g., NFS or ARP) and those which are used by network applications. These additional network applications need a third driver: `socket.sys`. `Socket.sys` implements the driver support, which thus allows use of the Berkeley 4.2BSD semantics of using `read()` and `write()` calls. FTP is an example of a program that uses the `socket.sys` library.

The network library is simply a file that an application program would link in with the executable, allowing that application full access to TCP/IP and ONC routines. The library includes the `socket`, `RPC`, `XDR`, and `NIS` routines one would find on SunOS. The one thing not available those `RPC` routines used to support server operations (e.g., the `portmapper`). The library routines would, in turn, call the appropriate device drivers for services.

One interesting design problem with the PC is that the Ethernet controllers can be quite primitive. The original `3c501` card, for example, could not receive back-to-back packets, since it couldn't receive a packet at the same time as it was making the first one available to the client. The client (the device driver) had to clear the buffer space out of the card before the card could receive new data.

Not receiving back-to-back packets is potentially a real problem. Often, if data is coming from an NFS host, it ends up being fragmented by the IP layer. A Unix 4096-byte block would end up being broken up into at least four different Ethernet packets. A second problem is in the case of broadcasts. Broadcasts are frequent, and receiving one may interfere with normal data flows.



For TCP transport users, there is a simple solution. Setting the maximum segment size to 1024 limits all data to one packet. Keeping the receive window size small means that the host won't be able to send more than one packet at a time.

In the case of NFS, which uses UDP as the underlying layer, the solution is not quite as easy. For NFS client operations, this means that large reads have to be broken into 1-kbyte chunks and requests for directory entries have to be limited to 1 kbyte at a time.

The problem is UDP-based applications can't always be controlled. What if the PC-NFS `showmnt` command is used to ask the server's mount daemon which systems are in its exported file systems? The mount daemon doesn't care about sizes: it just takes the list and sends it out. If the list is bigger than the maximum Ethernet MTU, fragmentation results, which means that the second fragment is lost, which in turn means that the RPC will time out. The solution, of course, is to get a more intelligent Ethernet card. From a historical perspective, it is interesting to see the hurdles that people had to go through to stuff a network into a PC.

### *PC-NFSD*

The PC-NFS daemon (PC-NFSD) is responsible for normal NFS operations but also has to handle two additional services:

- Authentication and file ownership/protection
- Local and remote print spooling

Authentication is necessary because the PC has no notion of users, file access controls, or any other of the issues any multiuser system would have automatically decided on. Local and remote print spooling are simply added services, making remote printers accessible to the PC. This makes PC-NFS comparable in the level of service to most of the PC-based local-area networks.

Authentication in the PC-NFS daemon is handled in several steps. First, the client must send a username and password to the daemon. The daemon will then authenticate this user to the local operating system. If accepted, the daemon returns a Unix-style UID and GID to the client.

This method is by no means foolproof when it comes to security. The username and password sent to PC-NFSD is encoded to prevent casual browsing. It is not encrypted in any sense of being difficult to break.

Obviously, this is not the most secure environment in the world. However, security and DOS don't necessarily go together very well. SPARCstations or any other modern workstation can be used where security is a requirement.

One useful feature of the PC-NFS daemon is that usernames that have been recently resolved can be cached. In the case of a very large `passwd.by-`



name map where users are sharing the same username, this makes a difference.

The canonical case for caching usernames is of course a class. It makes sense to give each PC the same username in this environment, since there is nothing to prevent any one from accessing any of the PC workstations.

Typically in such an environment, most of the NFS files are exported as read-only file systems. Software for the PCs and read-only data files can be kept there. Typically, the user is required to furnish a local disk to be used for results. This prevents one user from walking all over another's data.

### *Locking*

Locking for the PC has two problems. First, there are different kinds of locks available on the PC than in a Unix environment. Second, the PC has enough operational problems that adding a status monitor on top would be foolhardy. To address both of these things, the lock daemon was enhanced. In addition, the NFS daemon was changed to support additional functionality.

For PC-NFS record locking is crucial. In the more generic Unix environment, locking is advisory and few applications actually use it (they instead use some ad hoc or proprietary solution). In DOS the idea of controlled shared file access is a given: something that Unix ignores.

The lock daemon has two PC extensions. First, the daemon needs a provision for unmonitored locks. Second, additional RPC calls were added for DOS locking semantics. With an unmonitored lock, the server will not check to see if the client holding locks is still operational. Whenever a PC-NFS client mounts any file system on a server it frees all locks, thereby freeing any remnants from prior sessions. This is important because the PC maintains no lock state when it reboots, so it doesn't try to recover locks.

PC-NFS clients do maintain lock state through server reboots. The PC reestablishes contact with the lock daemon when the server comes back up. The semantics of locks were extended to combine the concepts of file sharing and access with file locking. File and recording locking is fairly similar to the Unix SVID locking. File access is a second indicator which determines if other users should be allowed a certain type of access. This was accomplished with the lock daemon (lockd) using a special byte 0 semaphore.

### *PC-NFSD Printing*

In addition to basic NFS operations, the PC-NFS daemon allows the user to spool files for remote printers. The client informs the PC-NFSD server of

the intention to spool print files. The server will return the path of an NFS-mounted directory where files can be written.

At that point, the PC-NFS application will intercept the print request and write that data to a spool file. When the user is done printing, the PC-NFS client informs the daemon that the file should be printed. The daemon then invokes some local printing mechanism, such as `lpr`, to print the file.

#### For Further Reading

Sun Microsystems, *SunNet License Administrator's Guide*, Part No: 800-3307-10, Revision A of January 1990.

———, *SunNet License Production Tool User's Guide*, Part No: 800-3308-10, Revision A of 19 January 1990.

———, *SunNet License Production Tool Administrator's Guide*, Part No: 800-3496-10, Revision A of 19 January 1990.

# Mail





## Overview

We saw in Chapter 9 in the discussion of the Domain Name System how a local system can, given a domain address, find the name of a remote host willing to handle mail for that user. This was the MX (for mail exchange) record.

Here we look at how the message is actually transferred. In the Internet world, the protocol used is the Simple Mail Transfer Protocol (SMTP). SMTP is simply a message transfer agent: a method of moving a message from one system to another.

In the Internet world, local delivery (or forwarding or rejection) is up to some program on the host. On a Sun Unix system, this is the domain of the sendmail mail handler. There are other mail handlers available. As long as the two handlers are able to speak SMTP, they can exchange a message.

There are other ways of transferring messages. In a strictly Unix environment, the Unix-to-Unix Copy Program (UUCP) is often used. The sendmail mail handler is able to “speak” both UUCP and SMTP when transferring messages.

There is yet a third approach, based on the X.400 standards. Here, an X.400 gateway is used to link the Unix-based systems to the broader, X.400 messaging environment. Often, X.400 is used as the glue that holds together many different local mail systems via the use of gateways.

## SMTP

We start with the Simple Mail Transfer Protocol, developed as one of the first Internet applications. What is ironic is that electronic mail was developed as something of an afterthought. Few realized that messaging would

become the single most important application—at the time the ability to remotely log into a host was considered the most pressing need.

Let us for the time being ignore the question of what a message looks like. How a message is formatted, where to put the recipient, the subject, the source of the message, and the message contents, are all irrelevant as far as the question of message transfer is concerned. The mailer, the local process that is in charge of mailing a message to a remote host, will extract the information necessary: usually just the name of the file that contains the message and who it goes to.

Once the mailer decides that it needs to send a message to another host, it opens up a TCP connection to the well-known port that has an SMTP-speaking process. At that point, a dialogue ensues between the sender and the receiver. Remember that the message may not have reached its ultimate destination: the remote system may well be a mail exchange which will take a message and forward it to yet another system.

Even if the message has reached its eventual destination, we don't really know how the remote mailer will do the delivery. Most systems have more than one user interface. In a Sun environment, for example, the user has the choice of a window-based application complete with a little mailbox that lifts a flag when mail is delivered or a command line-driven utility.

There are basically three steps to an SMTP exchange. First, the sender must be identified. Second, one or more recipients of the mail are identified. For each of the recipients, the remote system decides if it can accept mail for that person: if that recipient is on the remote system or if the remote system would know how to forward it to another host.

Finally, the actual data is sent. Notice that by identifying multiple recipients, a single copy of the data can be sent. For a message addressed to many people, this drastically cuts down on the amount of network bandwidth needed.

A typical dialogue would thus consist of:

```
MAIL FROM:<reverse-path>
```

```
250 OK
```

```
RCPT TO:<forward-path>
```

```
250 OK
```

```
DATA
```

```
345 INTERMEDIATE
```

    All subsequent lines sent are considered to be data.

    the end of data is signaled by a line with only a period.

```
250 OK
```

Notice that the replies to each of the three commands consist of a number and a text message. The number is for purposes of the programs; the text associated with it is an advisory message for any humans that happen to be listening.

Each of the three digits handles a different aspect of the reply. The first digit indicates if the command was successful or not. A 2, for example, means that a command was positively completed. The second digit indicates what the message refers to. A 5 says this is an answer pertaining to the mail system. The third digit allows messages to be specified with finer granularity. Thus, the reply of 250 means that the mail system is indicating a positive completion of the command.

Now we can look at another dialogue in more detail. This example comes from the SMTP description in RFC 821 and is shown in Figure 13-1. Notice that the remote system is not able to handle messages for all the users to whom the local system is attempting to send mail. However, just because one recipient is rejected doesn't mean that the whole message has to be rejected: the inability to find a particular user is fairly common in mail systems. The local system might try another remote system and, after several attempts, would decide the message is undeliverable.

In addition to the basic commands; MAIL, RCPT, and DATA, there are several other messages and options defined in SMTP (see Fig. 13-2). These

S: MAIL FROM:<Smith@Alpha.ARPA>

R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>

R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>

R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>

R: 250 OK

S: DATA

R: 354 Start mail input; end with  
<CRLF>.<CRLF>

S: intermediate data

S: intermediate data

S: <CRLF>.<CRLF>

R: 250 OK

SMTP Commands	
HELO	S: HELO USC-ISIF.ARPA R: 250 BBN-UNIX.ARPA
MAIL FROM:<reverse-path>	
RCPT TO:<forward-path>	
DATA	
RSET	
SEND FROM:<reverse-path>	Mail (display on terminal).
SOML FROM:<reverse-path>	Send or mail.
SAML FROM:<reverse-path>	Send and mail.
VERFY <string>	Verify name in mailing list.
EXPN <string>	Expand mailing list.
HELP [ space <string>]	
NOOP	No operation.
QUIT	
TURN	Reverse roles.
SMTP Architectural Limits	
Username	Maximum 64 characters.
Domain name	Maximum 64 characters.
Path (forward or reverse)	256 characters.
Command line:	512 characters including the carriage return/line feed.
Reply line:	Same.
Text line:	1000 charcters.
Recipients buffer:	No more than 100 recipients.

### 13-2 SMTP Commands and Limits

commands allow functions like verifying mailbox names, finding out who belongs to mailing lists, and even allowing two systems to change roles. Changing roles means that after sending all messages, a local system allows the remote system to deliver any messages it has.

The answer to a command consists of a reply code and some text. The text is advisory and is meant for human consumption. The numbers, however, are meant to guide programs. Replies are three digits, each digit conveying a different piece of information (see Fig. 13-3). This same theory of reply codes will be seen again when we look at the FTP protocol.

#### *Finding Users*

Often a remote mailer will not be able to deliver a message directly but will have better information than the local mailer on how one might go about



First Digit	
1	Positive preliminary reply: command accepted, but action held: send another command to indicate if you should continue or not. Note that SMTP does not have any commands that require this reply.
2	Positive completion reply.
3	Positive intermediate reply used for command sequence groups.
4	Transient negative completion: you can request the action again.
5	Permanent negative completion reply.
Second Digit	
0	Syntax.
1	Information (reply to status or help request).
2	connections (replies referring to the transmission channel).
3	Unspecified.
4	Unspecified.
5	Mail system.
Third Digit	
502	Unspecified action.
504	Command is implemented but parameter is not.

### 13-3 SMTP Reply Codes

doing the mailing. Take the example of a simple PC. The PC can function as a mail handler for the local domain that includes many different users. Usually, the PC would use some form of larger system as a mail exchange. Any local messages that can't be resolved would always be sent to that mail exchange. The mail exchange will know how to forward them on to the proper network.

When an SMTP link is set up between the PC and the mail exchange, the PC will attempt to send mail to the first recipient. SMTP allows the exchange to answer two different ways:

251 User not local; will forward to <forward-path>

551 User not local; please try <forward-path>

A forward path is simply a list of hosts to contact, in order, to deliver the message. The local system needs to find the first host in the forward path (in the case of the 551 message) and send to that host.

To send to a host, a few other utilities need to be involved. The local mailer would take the host desired and issue a Domain Name System query, specifying a lookup for the "A" resource record, the internet address of the target host. Once the IP address is determined, a TCP connection

request is sent to that host through the well-known service port used by SMTP.

Once that host is reached, it might make sense to verify that the host can in fact handle messages for that particular person. The `verify (VRFY)` command allows a system to send a username and get back a response verifying that the user can be reached via the remote system.

One more useful utility is the `expand (EXPN)` command. If the remote user is a mailing list, it may be useful to find out who is on the list. `EXPN` takes a string and returns a multiline response that includes at least a username and maybe full names. It is not uncommon to use the `mconnect` utility to connect to a mail host and expand mail aliases and lists in an SMTP environment. This is one of the few instances where a user interacts directly with an Internet protocol.

A simple example of the `VRFY` and `EXPN` commands is presented in RFC 821:

```
S: VRFY Smith
R: 250 Fred Smith <Smith@USC-ISIF.ARPA>
S: VRFY Gourzenkyinplatz
R: 553 User ambiguous
S: EXPN Executive-Washroom-List
R: 550 Access Denied To You.
```

Notice that `VRFY` accepts a human name and is thus a primitive way of finding a user's mailbox:

One more bell and whistle that is not often used is the `SEND` command, which posts a message on a user's terminal instead of into a local mailbox. Assuming the Unix `talk` utility was not enough, an overly dedicated (but not overly productive) programmer could open a TCP socket and send the SMTP `SEND` command to display messages on a screen. Variants of this exotic command allow the user to *send or mail* (`SOML`) or even *send and mail* (`SAML`). Needless to say, most SMTP work is confined to `DATA`, `MAIL`, and `RCPT` commands.

### *Paths*

There have been several references to a forward path, which is not quite the same thing as an address. Local databases are often used to construct a list of hosts that are necessary to use to move a message from one path to another.

Take the domain address `Joe@Three`. Consulting the local database, we find that we cannot directly reach host `Three`, since it is not accessible (or doesn't have an SMTP port). We are able to set up an SMTP path to host `One`, and we know from past experience that one can send the message on to host `Two`, which can deliver the message.

There is a difference between the route and the address. When a host receives a forward path, the host eliminates its own name, leaving the next host specified in the source route. Note that the name of the receiving host might not be the first on a received message. This just means that the host should consult its forwarding database to figure out how to get to the specified host.

Figures 13-4 and 13-5 show a typical example of some mail traffic. Figure 13-4 shows a session being initialized with a HELO command, followed by a message being sent. Figure 13-5 shows an SMTP session over a TCP-based link. Notice that the data is being acknowledged at the underlying TCP layer, allowing the SMTP application to assume that the data has been delivered in order.

## RFC 822

SMTP defines how to move a message from one point to another. All one needs to know is who the message is from and who it goes to. The message itself is just arbitrary data as far as SMTP is concerned.

What a message looks like internally is the domain of RFC 822, which discusses what headers a message has and what the body looks like. RFC 822 defines message formats for textual messages only; it is not well-suited to multimedia mail although extensions are being discussed in the IETF for different types of body parts.

In addition to identifying addressees, header fields serve a wide variety of purposes, useful to both machines and people. Most messages contain a message ID field, for example. When a reply is sent to a message, the header field "In-Reply-To" is inserted with the originating message ID, allowing the remote user interface to assemble different messages in a dialogue.

To many of us, such a header field is of no use. If you have a plain primitive user interface, you look at the message ID, ignore it, and move straight down to the subject field. If you have a fancy user interface, on the other hand, your program might be smart enough to go into a database and pull out the old message referred to in the In-Reply-To header and display it on the screen along with the new message.

RFC 822 defines only a few required headers (see Fig. 13-6). All the rest are optional. With all the headers, it is up to the user interface to decide if anything will be done with them. For example, one could easily include a Receipt-Requested header. However, many mail systems don't generate receipts: nobody programmed that capability. It would be up to the human to send a message back saying "I got your message." Many mail interfaces can be configured to ignore (not display) most control fields in a mail message.



SUMMARY	Delta T	DST	SRC	
78	1.3516	33333333333+DEC	0C85E6	SMTP C PORT=3291 HELO garden-bra
80	0.0141	33333333333+DEC	0C85E6	SMTP C PORT=3291 MAIL From:<@UM1
125	1.5532	33333333333+DEC	0C85E6	SMTP C PORT=3291 RCPT To:<wyk238
131	0.2132	33333333333+DEC	0C85E6	SMTP C PORT=3291 DATA<0D><0A>
134	0.2095	33333333333+DEC	0C85E6	SMTP C PORT=3291 Received: by ga
135	0.0024	33333333333+DEC	0C85E6	SMTP C PORT=3291 -----<0D><0A>
136	0.0016	33333333333+DEC	0C85E6	SMTP C PORT=3291 tors pay<0D><0A
137	0.0016	33333333333+DEC	0C85E6	SMTP C PORT=3291 are assigned to
139	0.0210	33333333333+DEC	0C85E6	SMTP C PORT=3291 not to be frame

DETAIL

SMTP: ----- SMTP data -----

SMTP: HELO garden-brau.csd.uum.edu<0D><0A>

SMTP:

Frame 78 of 1218

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 13-4 SMTP Session

SUMMARY	Delta T	DST	SRC	
16	0.1715	3Com 063885+Intrln002C60	SMTP C PORT=1534	Received: by li
17	0.0045	3Com 063885+Intrln002C60	SMTP C PORT=1534	r from you? Am
18	0.1191	Intrln002C60+3Com 063885	TCP D=1534 S=25	ACK=39934987
19	0.0099	3Com 063885+Intrln002C60	SMTP C PORT=1534	se tell me: it
20	0.3796	Intrln002C60+3Com 063885	TCP D=1534 S=25	ACK=39934998
21	0.0077	3Com 063885+Intrln002C60	SMTP C PORT=1534	ope Geo is well
22	0.2167	Intrln002C60+3Com 063885	TCP D=1534 S=25	ACK=39934998
23	1.9587	Intrln002C60+3Com 063885	SMTP R PORT=1534	250-Message acc
24	0.0363	3Com 063885+Intrln002C60	SMTP C PORT=1534	QUIT<0D><0A>

DETAIL

SMTP: ----- SMTP data -----

SMTP: 250-Message accepted and queued for delivery<0D><0A>...

SMTP:

Frame 23 of 156

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 13-5 SMTP Session



Field Name	Description
Return-path:	Return address.
Received:	
Received from:	Domain (sending host).
Received by:	Domain (receiving host).
Received via:	Physical path.
Received with:	Link/mail protocol.
Received ID:	Message ID.
Received for:	Address (original address spec). All received fields can have “; date-time” at end.
Reply-To:	Address.
From:	
Sender:	
From:	
Resent-Reply-To:	
Resent-From:	
Resent-Sender:	
Resent-Date:	
Date:	
To:	
Resent-To:	
CC:	
Resent-cc:	
Bcc:	Blind carbon copy
Resent-bcc:	
<b>Optional Fields</b>	
Message-ID:	
Resent-Message-ID:	
In-Reply-To:	
References:	
Keywords:	
Subject:	
Comments:	
Encrypted:	

**13-6**  
RFC 822 Header Fields

Header fields in messages have a rigidly fixed format. They consist of a field name, which is a string of characters, terminated by a colon. After the colon is some data specific to each field. In the basic RFC, there are several header fields defined. Users can add their own header fields (technically, the field name should start with X- as in X-Fruit-of-the-Day). In addition, several other standards document additional header fields. Privacy Enhanced Mail, for example, defines how to keep message integrity checks, credentials, and other security information in a message header.

There are also extension fields (those published as a formal extension to RFC 822), none of which have names beginning with X-. User-defined fields are also extensions but have not been published and may be preempted by published extensions.

### *Forwarding*

A variety of trace fields are added to messages as they are forwarded from one host to another. Most of these fields help define the return path for the message. It is important to have definitive information about the originator and a route back. Return-Path is the route back to the originator. Note that there is also a Reply-To that allow the user to specify either a better address than source routing can provide or to put another username in for to handle replies.

Received fields are added by each transport service as a trace. Each transport service that relays the message puts the received header in. The Via and With parameters may also be added; in fact there can be several With parameters to fully specify which protocols are used.

Figure 13-7 shows an example of a typical mail message. Notice the multiple received headers as the message moved forward. Notice that SMTP was used for the first forwarding but that another protocol, UUCP, was used for the second. In this case a program equivalent to the SunOS Vacation command is automatically replying. The well-behaved vacation program tries not to send automatic replies to distribution lists, but it has been known to happen that every delivery to a user by a mailing list results in a storm of replies to the entire list.

One more message of interest is included in Figure 13-8. Notice how often this message was forwarded as it moved from domain to domain. This message started out as a news bulletin on the Usenet network. The message reached ucbvax at Berkeley, where it was turned into a mail message addressed to the ISO Development (ISODE) mailing list.

The message was then fed into the Network Information Center, where the ISODE list actually resides. Isode@Nic.DDN.Mil is a user like any other, so the DNS MX records are consulted to find the appropriate host and internet address. A TCP connection is set up, and a protocol such as SMTP used to transfer the message.

From hqafsc-vax.af.millpostmaster"%hqeis.mrgate."a1.decnet  
Fri Dec 21 19:36:39 1990 remote from uunet  
Received: by malamud.com (UUL1.3#5653)  
from uunet with UUCP; Fri, 21 Dec 90 16:51:04 PST  
Received: from HQAFSC-VAX.AF.MIL by uunet.uu.net (5.61/1.14)  
with SMTP Id AA13352; Fri, 21 Dec 90 19:36:39 -0500  
Message-Id: <9012220036.AA13352@uunet.uu.net>  
Date: 21 Dec 90 19:32:00 EST  
From: "HQEIS::MRGATE::"A1::POSTMASTER"  
<uunet!hqafsc-vax.af.millpostmaster"%hqeis.mrgate."a1.decnet>  
Subject: Automatic reply to mail addressed to MCDANIEL  
To: "carl" <carl@malamud.com>

From: NAME: Mail Postmaster  
FUNC:  
TEL: <POSTMASTER AT A1 AT HQEIS>  
To: WINS%"CARL@MALAMUD.COM"@KRDEV@MRGATE

THE AFSC DDN PROGRAM MANAGER DUTY STATUS.

I will be on leave from 14 Dec 90 till 21 Dec 90.

I will return to duty on 24 Dec 90.

Emergency or Urgent DDN & AF Concentrator issues can be addressed  
to: Hq AFSC/SCXR  
Major McCoy  
DSN 858-7904  
Coml: 301-981-7904  
Email Address: MCCOY@HQAFSC-VAX.AF.MIL

Routine issues will be handled upon my return, so please leave a  
message or forward an email message.

NOTE: THIS IS AN COMPUTER SYSTEM AUTO REPLY MESSAGE AND IS SENT  
OUT EVEN TO MAILER DISTRIBUTION LISTS, TCP-IP, INFO-VAX, AND  
OWNER-IETF EMAIL MESSAGES.

Regards,

From NIC.DDN.MIL!isode-RELAY Wed Nov 21 20:58:15 1990  
remote from uunet  
Received: by malamud.com (UUL1.3#5653)  
from uunet with UUCP; Wed, 21 Nov 90 18:07:59 PST  
Received: from NIC.DDN.MIL by uunet.uu.net (5.61/1.14)  
with SMTP id AA09280; Wed, 21 Nov 90 20:58:15 -0500  
Received: from ucbvax.Berkeley.EDU by NIC.DDN.MIL  
with TCP; Mon, 19 Nov 90 16:51:49 PST  
Received: by ucbvax.Berkeley.EDU (5.63/1.42)  
id AA04161; Mon, 19 Nov 90 16:50:07 -0800  
Received: from USENET by ucbvax.Berkeley.EDU with netnews  
for isode@nic.ddn.mil (isode@nic.ddn.mil)  
(contact usenet@ucbvax.Berkeley.EDU if you have questions)  
Date: 13 Nov 90 07:31:21 GMT  
From: uunet!ucsd.edu!usc!zaphod.mps.ohiostate.edu!mips!cs.uoregon.edu!  
logicse!zephyr.ens.tek.com!tektronix!nosun!qi-clab!m2xenix!quagga!ucthpx!uctcs.uucp!gram (Graham Wheeler)  
Organization: Dept. of Computer Science, University of Cape Town  
Subject: ISODE - what is it; where can I get it?  
Message-Id: <887@ucthpx.UUCP>  
Sender: uunet!nic.ddn.mil!isode-relay  
To: isode@nic.ddn.mil

OK, I guess I'm dumb. All this time I've been killing articles on ISODE 'cos I haven't known what it was and haven't been too interested. Suddenly I put 2 and 2 together, and figured 'dev-environ == DE' Aha!

So now I'm interested, but I can't go back and read all the news I've killed. So at the risk of boring everyone, can someone give me a brief description of ISODE, and also tell me if it is possible to get it (from a mail server; I can't ftp from here). To spare others, please E-mail a response. Thanks.

Graham Wheeler	"Don't bother me,
Data Network Architectures Lab	I'm reading a 'Crisis'!"
Dept. of Computer Science	
	Internet <gram.uctcs@f4.n494.z5.fidonet.org>
University of Cape Town	
	BANG: <...uunet!ddsw!olsa99!uctcs!gram>



When mail messages are delivered to the mailing list program at Isode@Nic.DDN.Mil, the program there transfers them to another source address, isode-relay. Having mail come from a relay means that programs can automatically detect mailing list data (and not generate automatic replies).

From isode-relay, the message was burst and sent on its way. This particular version of the message went to a machine called UUnet, a commercial mail relay service in Virginia. From UUnet it went to the machine malamud.com, and then on to the user Carl.

One more point of interest. Notice the from line on the message, used to reconstruct a reverse path. This UUCP-style address indicates a path that could be used to reach this particular user. Each host in the path is separated from the others by the ! ("bang") character.

### *From, Sender, Reply-To*

There are several instances where the sender of a message may not be the person who is actually responsible for the message. A typical example is when a secretary sends a message from his account. Two additional fields help out here. The Sender field indicates who typed in the message, and the Reply-To field indicates who is handling replies:

```
From: Geory Jones <Group@Host>
Sender: Secy@host
Reply-To: Secy@host
```

Another use of these fields is when a single person sends a message, but answers should go to many different people. In this case, the Reply-To field might contain several people:

```
From: George Jones <Jones@host.Net>
Sender: Jones@Host
Reply-To: The Committee: Jones@Host.net,
          Smith@other.org,
          Doe@Somewhere-Else;
```

### *Message Encapsulation*

Normally, the message text is simply that: a series of arbitrary ASCII characters that mean nothing to the program (and may even mean nothing to the human using the program).

There is an instance when it may be necessary to batch several messages together: encapsulating the messages inside another message for transmission. Some mailing lists generate a very large amount of traffic. An example is the Sun-Spots mailing list, which is sent to hundreds of people (and

thus generates hundreds of messages). Having each message sent out to each person would generate an incredible amount of traffic.

Instead, the Sun-Spots list uses a digest format. Many different messages are all bunched up into a larger message (typically 15,000 to 18,000 bytes). When the digest is received, the user will want to be able to burst it and then respond to individual messages.

A very simple convention is used. First, there is a normal header section, corresponding to the large package. Inside that package are a series of messages, separated by an encapsulation boundary (EB), which is simply a line of dashes.

If a message really has a line of dashes in it (or has a dash at the beginning of the line) the first character is simply changed from a dash to a dash plus a space. During bursting of the message, any line that starts with a dash plus a space is not treated as an encapsulation boundary. Simple? Sure. That's the whole point.

### Using DNS to Forward Messages

The reader may recall the discussion of resource records in Chapter 9 on the Domain Name System. A special type of resource record is the MX record, which is used extensively for mail transfers.

An MX record indicates, for a given domain name, which system is to act as the mail exchange. In addition, there is a preference value. Several different MX records may exist for a single domain.

The preference number is the order the mailer should use to deliver to a particular host. The lowest numbered MX is the one to try first. There can be multiple records with the same priority.

Note that there are a few other resource records (RRs) that a mailer may choose to use. The canonical name (CNAME) RR says that the domain name queried for is actually an alias for another domain name.

The well-known service (WKS) RR stores information about networks services (e.g., SMTP) that a given domain name supports. Finally, given some host that is willing to act as a mail exchange, we of course need to consult the Address (A) record to find out the internet address for that host.

The primary requirement for routing messages is to prevent mail looping. The mailer on a host that is listed as an MX for the destination host may only deliver to an MX which has a lower preference count.

Another potential problem is incomplete answers from DNS. If an answer is incomplete, it may leave out a critical MX RR. Therefore, a mailer will only accept responses that have a complete answer. This means that a mailer which receives an answer with the truncation bit set should repeat the query using a virtual circuit.

Step 1 is for the mailer to issue a query for an MX RRs for the remote domain name. If the query is successful (no truncation bit and no error codes), the mailer will receive one of two resource records:

- A list of MX RRs
- A CNAME RR

If the CNAME is returned, the query is repeated until a list of MX records is presented. Once the mailer receives a list of MX RRs, the mailer will try to determine the appropriate host to handle this particular mail. If the list is empty, the mailer will act as if the MX is the remote host itself.

Otherwise, the mailer will look at the lowest preference value and issue an WKS query to see if that domain supports the mail service desired. This returns an IP address and port number, and the mailer then tries to establish the SMTP session.

Note that when a mailer goes through the list, it will remove any RR that doesn't support the desired service, based on the WKS field). If the mailer is in the list of RRs, it will remove any other hosts with a greater preference value.

After removing all these, the mailer may again end up with an empty list. This might be because nobody supports the desired mail service. Or, it could be that the domain system thinks this mailer is the the handler, but it really isn't. In either case, the message is undeliverable.

## UUCP

SMTP is one example of a mail transfer protocol. Another popular one is the Unix-to-Unix Copy Program (UUCP). UUCP preceded the Berkeley Unix support for SMTP and is still used for dial-up, demand-driven connections. UUCP does more than just transfer files: it allows a user on one unix system to send a work order to another to have certain things done and the results returned.

UUCP is really a whole collection of programs (see Fig. 13-9). The system is more than a little arcane but has the advantage of being widely implemented. We highly recommend the two "Nutshell" books published by O'Reilly and Associates for more information (listed at the end of this chapter).

UUCP works over point-to-point links, such as telephone lines or cables, or over networks, such as Ethernet (though it is more likely that the SMTP mailers will be used in LAN and WAN environments). A variety of programs allow the user to connect to another system, move files, and receive data. In addition, there are a variety of databases that include information on who the remote systems are, how to dial them, what they can do on your file system, and how often to poll.



Command	Description
cu	Connects a computer to a remote computer. Allows transfer of files or execution of commands on either computer without dropping the initial link.
uucp	Lets a user copy a file form one computer to another. UUCP creates work files and data files, queues the job for transfer, and calls the uucico daemon.
uuto	Copies files from one computer to a public spool directory on another computer (/var/spool/uucppublic/receive). UUCP lets you copy to any accessible directory on the remote computer: uuto tells the remote user to pick it up with uupick.
uupick	Lets user retrieve a file that is placed in /var/spool/uucppublic/receive (transferred in with uuto).
uux	Creates the work and data files needed to execute commands on a remote computer. Also adds a third file: the execute file which contains the command string to execute on the remote computer.
uustat	Displays the status of a requested transfer that was initiated with uucp, uuto, or uux. Lets you control queued transfers.
<b>Administrative Commands</b>	
uulog	Displays the content of the log file which has a record for each use of uucp, uuto, and uux.
uucleanup	Cleans up the spool directory.
uutry	Tests a call by invoking the uucico daemon.
uucheck	Checks to make sure you have the right directories, programs, and support files and looks for obvious syntactic errors.
<b>UUCP Daemons</b>	
uucico	Selects the device to use for a link, establishes the link, logs in, transfers data and execute files, logs results, and notifies the user by mail of transfer completions. The two uucico daemons talk to each other.
uuxqt	Executes remote execute requests. Searches the spool directory for execute files that have been sent from a remote computer. Opens it to get a list of data files. Checks to see if the files are available; checks the permissions file to see if you're allowed to do this; executes.
uusched	Schedules queued work in the spool directory. First randomizes the order in which remote computers are called, then invokes uucico.

### 13-9 UUCP Components

From this cursory description, it may be evident that the author views UUCP with some distaste. However, like many old systems, it serves a very valuable purpose. For many years, the Internet was sharply restricted to research organizations and large government bureaucracies.



Those people without a government contract still needed a way to exchange mail messages. UUCP served this purpose and still does. The author uses UUCP as a way of connecting his PC to the UUnet commercial mail exchange service.

UUCP forms the foundation for another valuable service, the Usenet. The Usenet is a very loosely organized bulletin board and news service. The basic idea is that a computer agrees to exchange messages with another computer. That computer in turn agrees to exchange messages with yet another.

With no formal routing and a loose mesh, it sometimes takes days to move a message from one side of the country to the other. At the same time, people on the other side of the country are originating their own messages. This means that every node has a slightly different view of the state of the bulletin board.

This system has a great asset: informality. Many sites continue to use UUCP links for news and messages at the same time as they use TCP/IP and SMTP to distribute mail locally and connect to the Internet.

The Sun mailer, sendmail, allows messages to be sent with both SMTP and UUCP as transport mechanisms (as well as any other mechanisms the user cares to add). Sendmail has some routing tables that instruct it which mail transport to use depending on the destination address required.

## Munging Messages

As we can see, there are several different message handling environments. The loose UUCP world is one, the Internet RFC 822 world is another. There are others: MCI Mail, Compuserve, and X.400. Each environment has a different format for messages.

Moving a message from one environment to another is called munging. Munging is increasingly becoming an issue as more and more systems get connected together via gateways. Electronic mail is almost always the first and most prevalent application that needs to work across the boundaries of different environments.

Munging used to be fairly simple. Systems had to be able to handle moving messages with different address formats but essentially the same type of text. This is because most headers were ignored: only information like the sender and the recipient caused an explicit action to occur.

An example of easy transformation was moving an Internet message into a UUCP domain. In the UUCP world, a mail address consists of an explicit path, separated by an exclamation point:

UUnet!Malamud!Carl

This address says to move a message from the host UUnet to the host Malamud and finally to the user Carl. A good Internet address, however, would be something like:

Carl@Malamud.Com

Looking in the DNS server for Carl@Malamud.Com would identify the fact that the host UUnet was the MX mail exchange for Malamud.Com.

Given this information, we can easily change the DNS domain address into a UUnet address. Since UUnet is the mail exchange, we start with it. Next, we add Malamud, finally Carl.

Much more complex munging is necessary when going to and from the X.400 world. This is because X.400 has a variety of service elements available: header lines that indicate some specific action should (must) take place.

### Case Study: Sun's Mail System

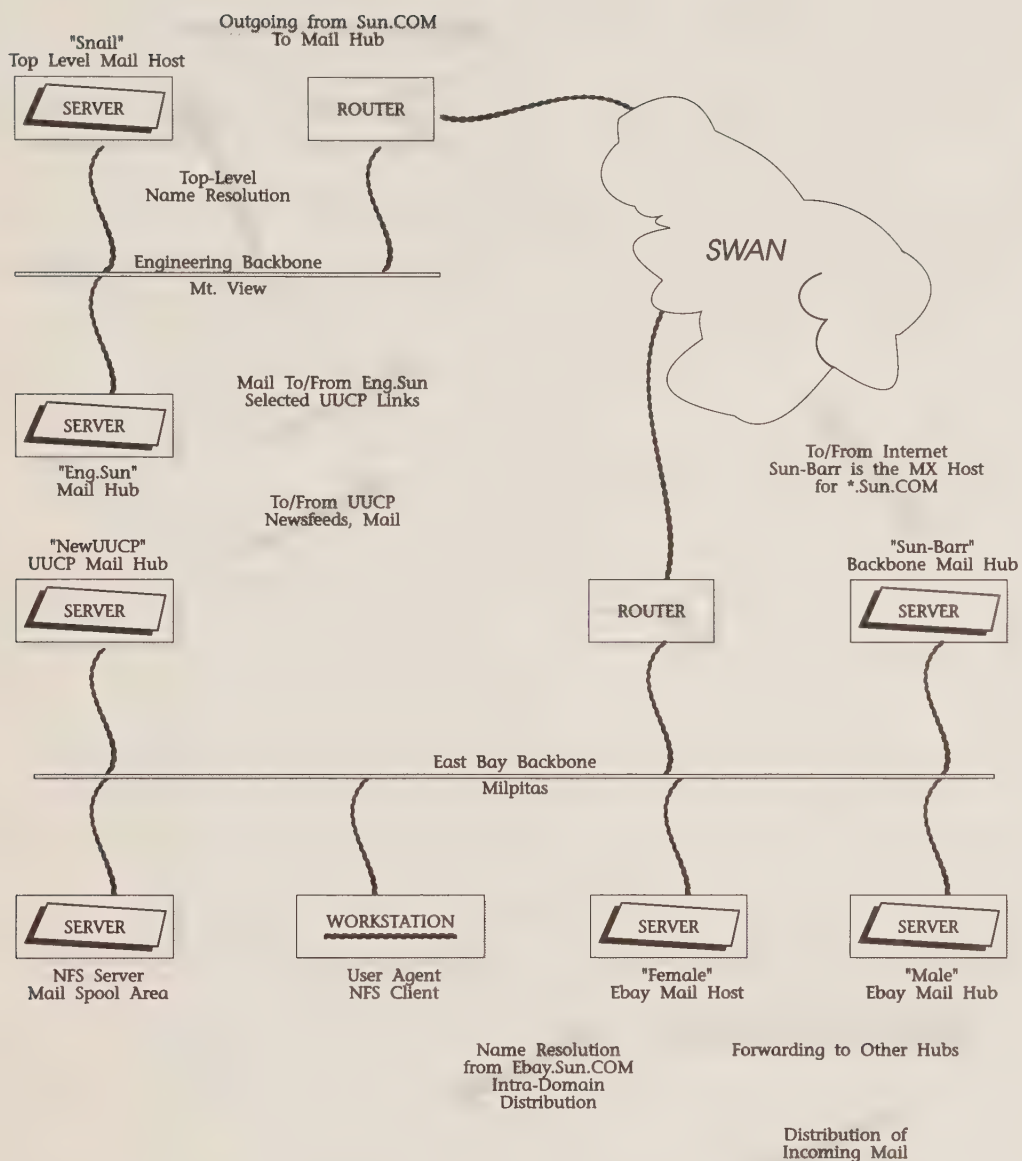
Before entering into the entrails of X.400, it might be useful to see how a real mail handling system is organized. Figure 13-10 shows a high-level overview of mail machines of interest for Sun's East Bay (Milpitas) subdomain. As the reader can see, there are a variety of methods for moving messages back and forth, and there are connections between the different message handling domains. The user, on the workstation, has some user interface program, such as the window-based mail tool. The mail tool is configured to look in a certain area for mail files, typically a delivery area, plus a storage area in the user's home directory.

Presumably, that mail lives on some host which is an NFS server. This is the place to which mail messages are ultimately delivered. These locations, spread throughout the Milpitas subdomain, all interact with two other hosts: "Male" and "Female."

Male is the mail hub for the East Bay and Female is the mail host for the East Bay domain. All the other domains have similar servers. The mail host is responsible for all outgoing mail messages.

Female, the mail host, handles name resolution. If a user addresses a message to "Jack," this is presumably a message within the domain EBay (Jack@EBay), and Female will resolve the name, find the area where Jack's messages are kept, and deliver the message.

On the other hand, the mail may be addressed outside the domain, say to Jill@Eng. Female will resolve the name as much as possible: we know that the message is for Sun's engineering domain. Other messages may not be resolved so easily. After doing as much resolution as possible, Female forwards the message to Male. Male is a mail hub: it moves all data going out one step forward to its final destination. If the message is going out via



## 13-10 Sun's Mail Servers

UUCP, it might go to one of two UUCP hubs, NewUUCP or Eng.Sun. Both these machines maintain UUCP-based relationships with a variety of hosts throughout the world.

Other messages will end up going out via the Internet. All outgoing mail to the Internet is handled by the host Sun-Barr. If a user is sending mail into the Sun domain from the Internet, it also goes out to this host.

Sun-Barr is the transition from the internal Sun network to the external Internet. Say you want to send messages to Jack@Sun.Com from some other place on the Internet (say Headhunter@OtherCompany.Com). The remote system would use the Domain Name System to look up the MX record for Sun.Com. This would indicate the host Sun-Barr.Sun.Com, with an IP address. An SMTP session would be set up and the mail transferred.

At Sun-Barr, a local database would be examined to find out who Jack was. Presumably, that search will find that the message is destined for Jack@EBay.Sun.Com. The Sun.Com part is really superfluous at this point, all we need to know is that Jack is in the East Bay and that the message goes to Male for ultimate delivery to Jack.

There is one more host of interest, and that is Snail. Snail is the top-level mail host on the Sun Wide-Area Network. This is where a message that can't be resolved is sent for the last attempt. This is an important function because the network has several different links to the outside world, and it is important to be able to move the message to the correct mail hub. Figure 13-11 shows an example of a message that has moved up through this hierarchy.

All these hosts are running a mail handler called sendmail. Sendmail looks at a message and decides who gets it next. If the message is at its destination, sendmail delivers it by inserting the message into the appropriate spool area.

If the message has not reached its destination, the message is analyzed to decide which mailer gets it next. A mailer in sendmail is a class of target mail hubs that have delivery characteristics in common.

Figure 13-12 shows statistics from four of the key sendmail programs on the Sun network. Notice that statistics are broken out by mailer. What a particular mailer does is defined in a file called sendmail.conf, the syntax of which is as obtuse as a government contract. While a full description of a particular mailer requires a detailed examination of the configuration file, some generalizations can be made. Male and Female have similar mailers defined. Messages are either delivered locally (using a simple NFS-based copy) or sent on.

The mailer SunUUCP handles the traffic going out via the UUCP protocols and will move the messages to either Sun.Eng or NewUUCP. The mailer Relay is any messages that can't be locally resolved or delivered. These messages are sent on to the host Snail, which will decide what to do



From EBay.Sun.COM!terryb Thu Jan 24 13:45:13 1991 remote from uunet  
 Received: by malamud.com (UUL1.3#5653)  
     from uunet with UUCP; Thu, 24 Jan 91 12:54:30 PST  
 Received: from Sun.COM by uunet.uu.net (5.61/1.14)  
     with SMTP id AA17654; Thu, 24 Jan 91 13:45:13 -0500  
 Received: from EBay.Sun.COM (male.EBay.Sun.COM) by Sun.COM  
 (4.1/SMI-4.1) id AA07745; Thu, 24 Jan 91 10:45:10 PST  
 Received: from female.EBay.Sun.COM by EBay.Sun.COM (4.1/SMI-4.1)  
     id AA26176; Thu, 24 Jan 91 10:45:06 PST  
 Received: from seescandy.EBay.Sun.COM by female.EBay.Sun.COM  
 (4.1/SMI-4.1-900117) id AA09681; Thu, 24 Jan 91 10:36:45 PST  
 Received: by seescandy.EBay.Sun.COM (4.1/SMI-4.1)  
     id AA02897; Thu, 24 Jan 91 10:43:08 PST  
 Date: Thu, 24 Jan 91 10:43:08 PST  
 From: uunet!EBay.Sun.COM!terryb (Terry Beyer)  
 Message-Id: <9101241843.AA02897@seescandy.EBay.Sun.COM>  
 To: carl@malamud.com  
 Subject: Mainframe Printing on Sun Printers Summary  
 Cc: terryb@EBay.Sun.COM

Carl -

The attached is a sample weekly summary of the volumes printed  
 from our mainframes on to Sun based printers.

Please let me know if you have questions.

Terry

—— Begin Included Message ——

From sunprint@irca8 Wed Jan 2 13:58:29 1991  
 Date: Wed, 2 Jan 91 14:00:31 PST  
 To: martyto@mock, proman@doubleplay, rf@zeppelin, robg@alps, so-  
 lan@sunnyskies,  
     stevek@wonder, terryb@seescandy  
 Subject: SUNPRINT Weekly Summary

#### SUNPRINT WEEKLY SUMMARY

From Previous Wed 14:00 to Wed Jan 2 14:00:04 PST 1991

PRINTER	REQUESTS	AVG BYTES	MIN BYTES	MAX BYTES
---------	----------	-----------	-----------	-----------

Mailer	Msgs From	Bytes From	Msgs To	Bytes To
<b>Mail Statistics for Female</b>				
Mail statistics from Tue Jan 22 00:02:16 1991 to Tue Jan 22 23:59:48 1991				
0 local	127	526K	96	2K
2 ether	12122	76413K	25139	100439K
5 relay	909	3795K	31788	144097K
9 sunuucp	2	2K	1303	7929K
Total	13160	80736K	58326	252467K
<b>Mail Statistics for Male</b>				
Mail statistics from Tue Jan 22 00:00:23 1991 to Tue Jan 22 23:59:44 1991				
0 local	6078	39803K	434	19K
2 ether	1886	15473K	17995	63529K
5 relay	8059	41480K	38387	157190K
9 sunuucp	2	3K	1291	7946K
Total	16025	96759K	58107	228684K
<b>Mail Statistics for Snail</b>				
Mail statistics from Tue Jan 22 00:00:06 1991 to Wed Jan 23 00:00:03 1991				
0 local	600	2004K	1948	3193K
1 prog	17	184K		
2 ether	771	3340K	120060	536758K
4 sales	137	1318K	804	6210K
5 swan	14	58K	143	562K
6 newswan	11081	62335K	15847	68819K
8 internet	2246	8758K	3259	23552K
Total	14849	77813K	142078	639278K
<b>Mail Statistics for Sun-Barr</b>				
Mail statistics from Tue Jan 22 04:05:06 1991 to Wed Jan 23 04:04:57 1991				
0 local	223	863K	2	2K
2 ether	8	25K		
3 ddn	11454	61641K	19628	93185K
4 fakeuucp	20	106K	5	17K
5 sales	135	540K	838	3344K
6 relay	21	46K	4092	29690K
7 area	4011	31713K	5825	17868K
8 uucp	588	2122K	1191	5504K
10 badarea	1952	14941K	2992	12507K
11 unknown	68	163K	2989	8586K
Total	18472	112135K	37570	170728K

with them. Ether is the mailer that handles delivery within the confines of an Ethernet.

Hosts Snail and Sun-Barr have significantly more complex mailers due to their central role as a hub. The Sales mailer is a special mailer that originated in the need to maintain UUCP-based links to sales offices. DDN is the mailer that handles all outgoing messages.

Two interesting mailers on Sun-Barr are Badarea and Unknown. Badarea is for badly formed addresses. Many people have hardcoded host names in their address: Jack@eng301c.Eng.Sun.Com. This hardcoding may get the message through, but has some potential problems. What happens when the user moves to another host?

While the names in the Badarea handler can be resolved, some contain baffling addresses. An incoming message might be addressed to Jack@Sun.Com, but we have no idea who the user Jack is. Or, a message goes out addressed to Bob@NonHost.TheMoon. Consulting our tables, we see no way to reach the top-level domain TheMoon, so we issue a DNS query. We find that we have no way of finding the handler for this domain, so the message is piped out to /dev/null.

The machine NewUUCP keeps links going with a wide variety of different hosts. Many of these are Sun sales offices, customers, or even just organizations who have agreements with Sun for the exchange of messages or news. Figure 13-13 shows a small sample of the UUtraffic command for this host.

## X.400

X.400 structures a collection of message transfer agents and user agents into a management domain—a collection of resources under a single administration. A domain maintained by an International Telecommunications Union (ITU) member (or registered private member) is an Administration Management Domain (ADMD). All other domains are Private Management Domains (PRMD).

The PRMD may be connected to multiple ADMDs, but for the purpose of any one message the PRMD is connected to a single ADMD. Routing for a PRMD consists of giving the message to the ADMD.

Routing for the ADMD is simple for a directly connected domain. For further routing decisions, the process is outside the specification, but note that a domain can easily be “directly” connected to another through the use of the OSI Session, Transport, and Network layers. The X.500 directory is used to find the address of a relevant domain.

An X.400 domain is composed of message transfer agents and user agents. The message transfer agents combine to form the message handling system. In addition, there are two other specialized components, such as message stores, access units, and distribution lists.



UUtraffic for NewUUCP									
Remote Host	Kbytes			Hours			Avg CPS		
	Recv	Sent	Total	Recv	Sent	Total	Recv	Sent	Total
varsun	23249.4	4685.2	27934.6	17.9	3.4	362	385	2014	1088
lsil	8295.1	14422.4	22717.5	2.6	2.9	870	1395	2622	5348
emsca	2601.3	11634.7	14236.0	7.8	29.0	93	111	2278	3554
teamone	30.5	11390.0	11420.5	0.0	14.3	175	222	36	152
enera	5950.7	3098.5	9049.2	3.7	3.4	449	255	210	1246
valid	693.9	7882.8	8576.7	0.3	1.5	555	1466	438	3290
xopusw	5654.8	2129.8	7784.6	3.3	0.4	476	1513	1746	546
cadence	2876.0	3404.9	6281.0	0.8	0.8	955	1203	258	246
sunisv	451.4	3425.2	3876.6	0.6	4.3	202	220	194	2119
iis	65.4	2423.1	2488.5	0.2	5.8	82	116	112	369
svmsg	1583.8	375.7	1959.6	2.6	0.8	168	125	648	198
ntrlink	7.2	1207.7	1214.9	0.0	0.2	571	1724	4	923
sunuru	22.4	1109.7	1132.1	0.0	1.8	144	172	30	158
poseur	316.0	520.7	836.7	0.2	0.1	461	1663	262	313
larse	32.4	802.1	834.5	0.0	1.0	191	220	14	24
synergy	7.0	474.2	481.2	0.0	1.2	91	110	12	14
soland	60.5	390.2	450.6	0.8	3.9	21	28	58	160
ufrt	156.1	265.8	421.8	0.5	0.5	88	155	228	290
wesfdd2	35.3	332.1	367.5	0.1	0.4	183	251	32	196
barcode	165.1	53.0	218.1	0.1	0.0	388	1795	116	48
SUMMARY									
Total active uucp sites:								31	
Total files recv:								11,426	
Total files sent:								20,632	
Total files:								32,058	
Total hours recv:								41.8	
Total hours sent:								76.3	
Total hours:								118.1	
Total Kbytes recv:								52,399	
Total Kbytes sent:								70,586	
Total Kbytes:								122,985	

13-13 UUCP Statistics for "NewUUCP"



The message store is simply a staging point that picks up messages from the message transfer system and then holds them for the user. The user can submit messages either directly to the message transfer system or have them staged at a message store.

The distribution list is a set of addresses, which gets translated into individual messages. The access unit is a gateway to another message handling environment. X.400 defines access units for interconnection to telex, postal delivery, and a few other environments. The DEC X.400 gateway is an example of an access unit.

The message store appears as a user agent to the message transfer agent: it accepts delivery of messages and then holds them for the user. In addition, the user agent can submit messages to the message store, which then hands them off to the message transfer agent. An example of this scenario is a PC (the user agent) and a VAX operating as the post office (the message store).

Message stores can be simple stores, as the name implies. Additional services can also be built into the message store, such as auto-forwarding of messages on the behalf of a user.

## *Messages*

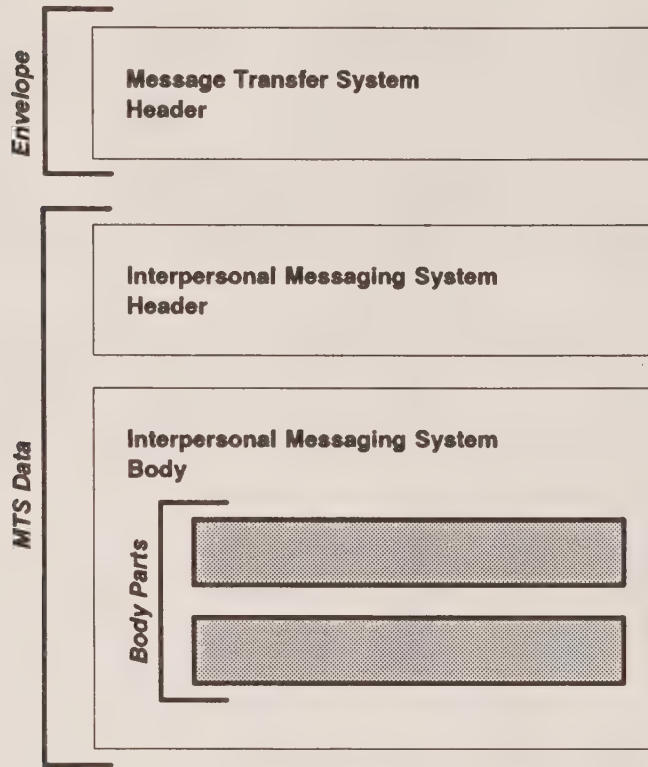
X.400 breaks a message up into three pieces (see Fig. 13-14). Each message has a header, put in by the message transfer system. The rest of the message is considered data by the message transfer system.

Certain classes of user agents, those compatible with the Interpersonal Messaging Service (IPMS), are able to add further structure to the message. This layer, similar to an upper-level protocol, also consists of a header and a body. There can be several different body parts in an IPMS message. Each body part can be structured in a different format (e.g., one piece as straight ASCII text, another part as a graphic image). Multiple body formats in one message is one of X.400's strongest features, providing direct support for multimedia mail.

## *Naming*

A user of the X.400 service is known as an originator/recipient address (O/R address). This user could be an individual user or a distribution list. It is also possible to use an X.500 directory name, which will then return an X.400 address.

As with network-layer addressing, the goal is to provide each user with a unique address. As with the OSI network addressing standards, a variety of different formats are defined (see Fig. 13-15). There are four different address formats defined. For each, there are a variety of different elements that can identify a user. Some elements are mandatory and others are con-



**13-14**  
Components of an  
IPMS Message

ditional—their use is up to a particular domain. For the postal address, for example, the country name and postal code are required elements. The name of the physical delivery service is optional—in the United States this information might specify Federal Express, UPS, HDL, or maybe even the United States Postal Service. In countries with a monopoly, the field would not be necessary.

## IPMS

The Interpersonal Messaging Service (IPMS) is a subclass of user agents. The IPMS defines a set of messages with a specific structure. Instead of some arbitrary message, the message is structured as a header and a set of body parts.

As with the address format, the IPMS includes a variety of different elements. Some elements are considered basic and are thus part of every implementation (see Fig. 13-16).

A basic service, for example, is a content type indication. The content type indication is a field in the IPMS header that indicates what the content looks like. A variety of timestamps are also provided. Finally, the user must be able to register its capabilities for any of the optional services.

Attribute Type	O/R Address Forms			
	Mnemonic	Numeric	Postal	Terminal
<b>General</b>				
Administration domain name	M	M	M	C
Common name	C			
Country name	M	M	M	C
Network address				M
Numeric use identifier		M		
Organization name	C			
Organizational unit names	C			
Personal name	C			
Private domain name	C	C	C	C
Terminal identifier				C
<b>Postal Routing</b>				
Physical delivery service			C	
Physical delivery country name			M	
Postal code			M	
<b>Postal Addressing</b>				
Extension postal address O/R address components			C	
Extension physical delivery address components			C	
Local postal attributes			C	
Physical delivery office name			C	
Physical delivery office number			C	
Local postal attributes			C	
Physical delivery office name			C	
Physical delivery office number			C	
Physical delivery organization name			C	
Physical delivery personal name			C	
Post office box address			C	
Post restante address			C	
Street address			C	
Unformatted postal address			C	

Attribute Type	O/R Address Forms			
	Mnemonic	Numeric	Postal	Terminal
Unique postal name			C	
Domain-defined	C	C		C
M - Mandatory C - Conditional				
Note: Postal address can be unformatted (all information in one attribute). This table applies to formatted postal addresses. The unformatted address consists of the <i>unformatted postal address</i> attribute.				
Source: CCITT Blue Book Rec. X.402 (p. 121)				

## 13-15 (Cont.) X.400 Addresses

Basic Services	Access management
	Content type indication
	Converted indication
	Delivery timestamp indication
	IP-message identification
	Message identification
	Nondelivery notification
	Original encoded information types indication
	Submission timestamp indication
	Typed body
	User/UA capabilities registration
Source: CCITT Blue Book VIII.7 - X.400 (p. 38)	

## 13-16 Basic IPMS Components

In addition to the structure of the message, the IPMS includes notifications. The message transfer service gives delivery and nondelivery notifications. The IPMS adds to this to include receipts and nonreceipts (of course, the user agent providing the IPMS may decide not to provide a nonreceipt or receipt notification).

There are many different optional elements defined in the IPMS. These options, if supported, typically consist of a field in the IPMS header. Figure



13-17 shows the different services offered. The services are categorized as being provided on the origination or reception of a message. In each of these categories, they are further categorized as being essential or additional. An essential element is one that must be provided by the IPMS, but the user doesn't have to choose it. An additional element is up to each implementation (or management domain) to decide if it should provide it.

An easy example of an additional element is the additional physical rendition, which allows the user to have a copy of the message printed. This is optional on both sides. The auto-forwarded indication is an example of an element that is additional for the message sender: there is no requirement that you will be notified if a message you sent was automatically forwarded to yet another user. On the other hand, this element is considered essential on reception: a user receiving a message should be able to find out if the message was in fact auto-forwarded.

There are several elements that are not available on a per-message basis but must be contracted for (see Fig. 13-18). For example, the message transfer system can hold a message when it receives it, delivering it later. Implicit conversion, redirection, restricted delivery, and services of this sort require additional resources within the message transfer system.

The list of possible IPMS elements is shown in (close to) its entirety to show the richness of the X.400 messaging environment. It should be noted that not all user interfaces will know what to do with all these elements. The underlying IPMS supports the functions, but the user interface will typically ignore additional elements. For this reason, a user may occasionally examine a message with a plain ASCII editor to see if there are additional elements included but not shown to the user.

## Message Transfer Agents

There are three types of objects in the message transfer system:

- Messages
- Probes
- Reports

A probe goes from one user to the message transfer agent just short of another user and is used to test connectivity. A report is an object sent from the message transfer system to a user. Each report concerns a message or probe.

Figures 13-19 and 13-20 show the structure of the message transfer agent modules. Incoming messages are delivered to either the main or report modules, which results in either a message, report, or probe being sent out.

The main module provides a variety of different functions. Splitting, for example, takes a single message with multiple addressees and splits it into

Element	Origination	Reception
Additional physical rendition	A	A
Alternate recipient allowed	A	A
Authorizing users indication	A	E
Auto-forwarded indication	A	E
Basic physical rendition	A	E
Blind copy recipient indication	A	E
Body part encryption indication	A	E
Content confidentiality	A	A
Content integrity	A	A
Conversion prohibition	E	E
Conversion prohibition in case of loss of information		N/A
Counter collection	A	E
Counter collection with advice	A	A
Cross-referencing indication	A	E
Deferred delivery	E	N/A
Deferred delivery cancellation	A	N/A
Delivery notification	E	N/A
Delivery via Bureau fax service	A	A
Designation of recipient by directory name	A	N/A
Disclosure of other recipients	A	E
Distribution list expansion indication	N/A	E
Distribution list expansion prohibited	A	A
Express mail service	A	E
Expiry date indication	A	E
Explicit conversion	A	N/A
Forward IP-message indication	A	E
Grade of delivery selection	E	E
Importance indication	A	E
Incomplete copy indication	A	A
Language indication	A	E
Latest delivery designation	A	N/A
Message flow confidentiality	A	N/A

Element	Origination	Reception
Message origin authentication	A	A
Message security labeling	A	A
Message sequence integrity	A	A
Multidestination delivery	E	N/A
Multipart body	A	E
Nonreceipt notification request indication	A	E
Nonrepudiation of delivery	A	A
Nonrepudiation of origin	A	A
Nonrepudiation of submission	A	A
Obsoleting indication	A	E
Ordinary mail	A	E
Originator indication	E	E
Originator requested alternate recipient	A	N/A
Physical delivery notification by MHS	A	A
Physical delivery notification by PDS	A	E
Physical forwarding allowed	A	E
Prevention of nondelivery notification	A	N/A
Primary and copy recipients indication	E	E
Probe	A	N/A
Probe origin authentication	A	A
Proof of delivery	A	A
Proof of submission	A	A
Receipt notification request indication	A	A
Redirection disallowed by originator	A	N/A
Registered mail	A	A
Registered mail to addressee in person	A	A
Reply request indication	A	E
Reply IP-message indication	E	E
Report origin authentication	A	A
Request for forwarding address	A	A
Requested delivery method	E	N/A
Return of content	A	N/A

Element	Origination	Reception
Sensitivity indication	A	E
Special delivery	A	E
Stored message deletion	N/A	E
Stored message fetching	N/A	E
Stored message listing	N/A	E
Stored message summary	N/A	E
Subject indication	E	E
Return of undeliverable mail	A	E
Use of distribution list		A
Source: CCITT Blue Book VIII.7 - X.400		

**13-17 (Cont.)** Optional IPMS Services

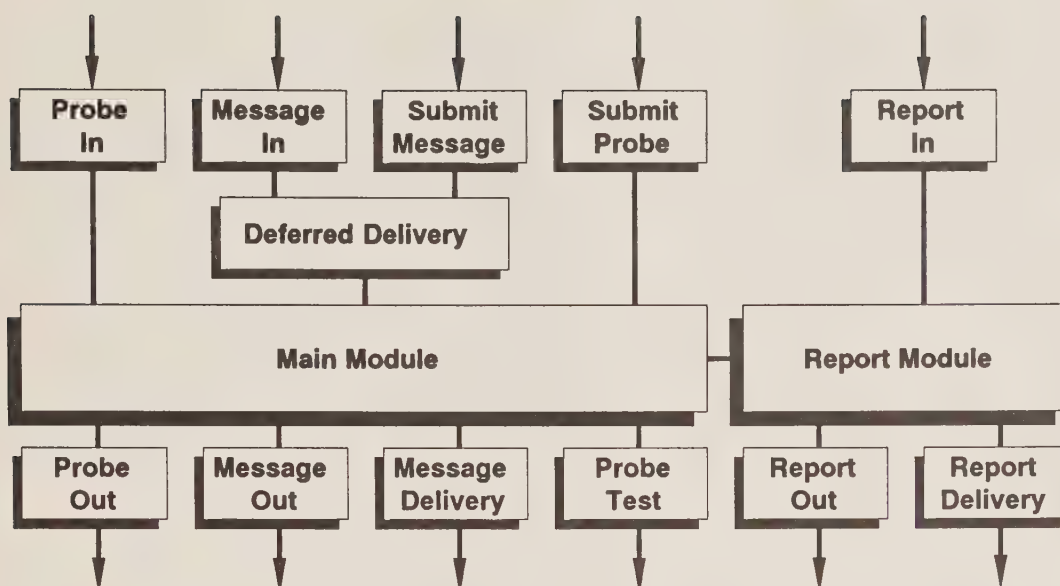
Alternate recipient assignment
Hold for delivery
Implicit conversion
Redirection of incoming messages
Restricted delivery
Secure access management
Stored message alert
Stored message auto-forward

**13-18** IPMS Contractual Facilities

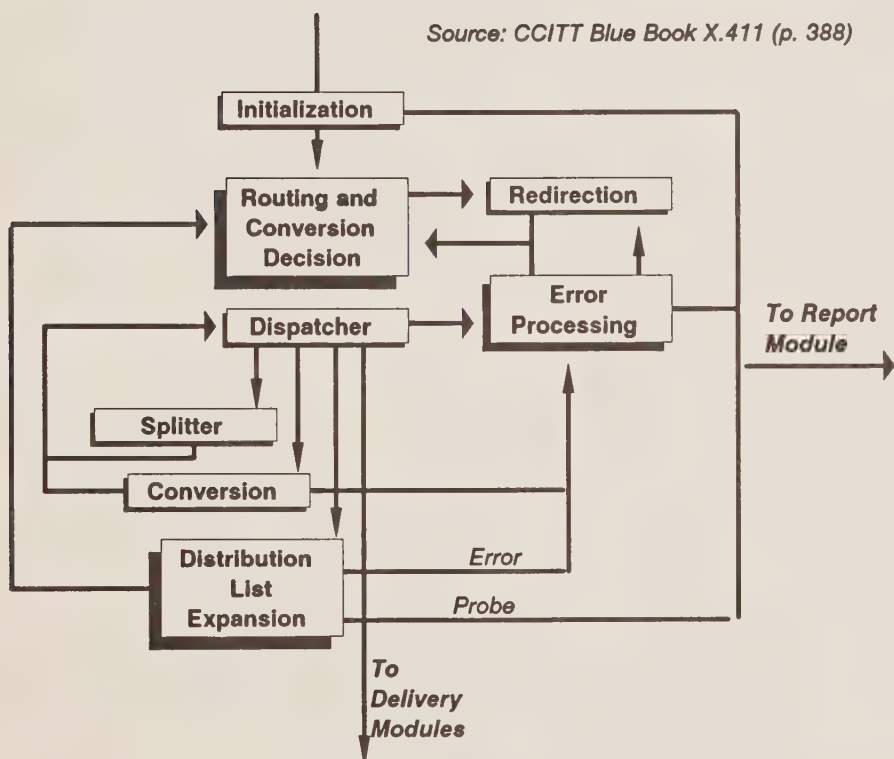
several messages. This happens when a split occurs: one recipient is located in one direction, and other users are in another direction. Joining is an optional step that joins multiple messages headed for the same place. For example, ten nondelivery messages from a single distribution list might be joined.

Name resolution, distribution list expansion, redirection, and conversion are all optional services. Nondelivery is an example of a message sent back





13-19 Components of the MTA



13-20 Main Module Components

to a user when a message is discarded, usually because the destination user or domain is unknown.

Finally, there is the question of routing selecting which message transfer agent (MTA) gets the message next. How information gets routed is up to a specific environment. Presumably, MTAs will use a hierarchical routing system: all unknown messages go to a “root” server, which is connected to other management domains.

Figure 13-21 shows the types of services offered by MTAs. Again, they are split up into basic elements (always provided), essential elements (available but not required), and additional elements (optional per implementation). Most of these elements of service are transmitted in the MTA header. For example, a user could specify that an alternate recipient is allowed (or not allowed). Deferred delivery allows a message to be submitted to the MTA with the proviso that it not be delivered before a specific time.

The deferred delivery cancellation is an element which shows that not all requests are always granted. A user can send a message with deferred delivery, say a press release announcing that DECnet Phase V has finally been shipped.

Once the message is sent out, it stays in a queue someplace in the message transfer system awaiting delivery. The originator may then decide, say because of technical difficulties, to delay the announcement and sends out a deferred delivery cancellation. There is no guarantee that the deferred delivery cancellation will catch up with the deferred delivery message in time to cancel the original message.

Figure 13-22 shows a typical X.400 message exchange. First, the sending node establishes the presentation-level connection and it is accepted. Next, the session layer activity start is sent. The X.400 P2 protocol is being invoked; P2 is the code for the interpersonal messaging service. Notice that this particular message is using the 1984 version of the X.400 protocols.

Figure 13-23 shows the P1 protocol (MTA to MTA) in more depth. This particular message is a delivery report. The message indicates a message identifier (the message Protocol Data Unit ID), as well as the originator of the report. It then includes trace information indicating when the message reached different locations.

In the content section, the identification of the original message is shown. The message was sent from the country United States, the administration management domain ATTMail, the private management domain called Retix, and from a particular user ID. The message also indicates that the messages was received by a user with the personal name Elena Seifrid and was delivered July 26, 1988.

Figure 13-24 shows the IPMS layered inside the message transfer protocol. The MTA header includes the originator of the message and the basic

<b>Basic Elements</b>	Access management
	Content type indication
	Converted indication
	Delivery timestamp
	Message indication
	Nondelivery notification
	Original encoded information types indication
	Submission timestamp
	User/UA capabilities registration
<b>Essential Elements</b>	Alternate recipient allowed
	Deferred delivery
	Deferred delivery cancellation
	Delivery notification
	Disclosure of other recipients
	Distribution list expansion history indication
	Grade of delivery selection
	Probe
	Requested delivery method
<b>Additional Elements</b>	Alternate recipient assigned
	Content confidentiality
	Content integrity
	Conversion prohibition if information lost
	Designation of recipient by directory name
	Distribution list expansion prohibited

### 13-21 MTA Elements of Service

information type: IA5text being a basic text message. The content type indicates that this is an IPMS message instead of some private protocol.

The message also includes various per message flags. Notice that the disclose recipients flag is on, allowing each user to know about the other users. No alternate recipient is allowed: if the message can't be delivered, it is dropped.

Next, there is information for the two recipients of the message, including an address and flags indicating that basic reports are requested. This is followed by trace information, indicating where the message has been so far. This message appears to have been relayed at least once by an MTA called VAX.

After all this information is the the IPMS header. The message has a unique ID, an originator, and two primary recipients. The reply requested flag is set to false. Of course, even if the flag were set true, there is no way

**SUMMARY** — Delta T — DST — SRC

	Delta T	DST	SRC	Description
4		Sun	0035CF+Intrln0061B1	ISO_PR Connect P1
6	0.0249	Intrln0061B1+Sun	0035CF	ISO_PR Accept
7	0.0127	Sun	0035CF+Intrln0061B1	ISO_SS Give Tokens, Activity Sta
9	0.0082	Sun	0035CF+Intrln0061B1	X.400 P2:Message US.ATTMAIL.reti
13	0.0110	Sun	0035CF+Intrln0061B1	ISO_SS Give Tokens, Activity End
17	0.0889	Intrln0061B1+Sun	0035CF	ISO_SS Prepare
19	0.0065	Intrln0061B1+Sun	0035CF	ISO_SS Please Tokens, Major Sync
21	0.2577	Sun	0035CF+Intrln0061B1	ISO_SS Finish
23	0.0188	Intrln0061B1+Sun	0035CF	ISO_SS Disconnect

**DETAIL**

ISO\_PR: ----- ISO Presentation Layer -----  
 ISO\_PR:  
 ISO\_PR: PPDU type = Connect Presentation (length = indefinite)  
 ISO\_PR: Mode selector = 0 (X410-1984 mode)  
 ISO\_PR: Checkpoint size = 5  
 ISO\_PR: Window size = 6  
 ISO\_PR: Dialogue mode = 0 (Monologue)  
 ISO\_PR: Application protocol = 1 (P1)  
 ISO\_PR:

Frame 4 of 171  
 Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev Frame 8 Next Frame 10 New capture

**13-22 X.400 Traffic**

the users could be forced to respond. The body of the message, consisting of a single body part, is composed of IA5Text. The message begins “Testing for Andre” and has an additional 1720 bytes.

### Distribution Lists

A distribution list is a list of individual O/R addresses or other distribution lists. In addition to its members, a distribution list has several other properties:

- The submit permission says which users and distribution lists can make use of this list.
- The expansion point is the O/R address of the list: the place where it is translated into its constituent members.
- An owner of the list.

A user sending mail to a destination doesn’t have to know that the destination address is a distribution list. Remember, however, that message charges can rapidly mount if one message turns into 100. One option in X.400 is to prohibit distribution list expansion.

The expansion point for a distribution list is an MTA (since the user does not exist). The expansion point will “burst” the message into several pieces,



```

DETAIL
X.400: ----- X.400 Message Transfer Protocol (P1) -----
X.400:
X.400: MPDU type = Delivery Report (length = indefinite)
X.400: Envelope:
X.400: Report MPDU identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/, SUN Tue Jul
X.400: Originator: /C=US/ADMD=ATTMAIL/PRMD=RETIX/0=UAX/PN=SEIFRID.ELENA.D/
X.400: Trace information:
X.400: Global domain identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/
X.400: Arrival = 26 Jul 1988 14:54:43-0700
X.400: Action = 0 (Relayed)
X.400: Internal trace info:
X.400: MTA name = SUN
X.400: Arrival = 26 Jul 1988 14:54:43-0700
X.400: Action = 0 (Relayed)
X.400: Content:
X.400: Original MPDU identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/, UAX 880726
X.400: Intermediate trace information:
X.400: Global domain identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/
X.400: Arrival = 26 Jul 1988 14:53:58-0800
X.400: Action = 0 (Relayed)
X.400: Content:
X.400: Original MPDU identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/, UAX 880726
X.400: Intermediate trace information:
X.400: Global domain identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/
X.400: Arrival = 26 Jul 1988 14:53:58-0800
X.400: Action = 0 (Relayed)
X.400: UA content id = 880726 14:53:53
X.400: Reported recipient info:
X.400: Recipient: /C=US/ADMD=ATTMAIL/PRMD=retix/0=sun/PN=seifrid.elena/
X.400: Extension identifier = 2
X.400: Per recipient flag = D0
X.400: 1... .... = responsibility flag on
X.400: .10. .... = confirmed report request
X.400: ...1 0... = confirmed user report request
X.400: Intended recipient last trace information:
X.400: Arrival = 26 Jul 1988 14:54:43-0700
X.400: Delivery = 26 Jul 1988 14:54:43-0700
X.400: Type of UA = 1 (Private)
X.400:

```

Frame 41 of 171

Use TAB to select windows

1 Help	2 Set mark	3 Zoom out	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	------------	---------	-------------------	--------------	--------------	----------------

### 13-23 X.400 P1 Protocol

sending them back out the message transfer system (see Fig. 13-25). Any charges for the activity would typically go back to the original user (although it is possible they would be incurred by the owner of the distribution list).

Distribution lists can be nested. When a nested distribution list is expanded, the ID of the parent distribution list, not the originator of the message, is used to check if expansion is allowed. A potential problem with distribution lists is recursion. To prevent this, a message includes a chain

```

DETAIL
ISO_SS: ----- ISO Session Layer -----
ISO_SS:
ISO_SS: Multi-frame TSDU: frames 9, 11, 12
ISO_SS: SPDU type = 1 (Give Tokens)
ISO_SS: SPDU type = 1 (Data Transfer)
ISO_SS:
X.400: ----- X.400 Message Transfer Protocol (P1) -----
X.400:
X.400: MPDU type = User (length = indefinite)
X.400: Envelope:
X.400: MPDU identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/, UAX 880726 14:53:53
X.400: Originator: /C=US/ADMD=ATTMAIL/PRMD=RETIX/0=UAX/PN=SEIFRID.ELENA.D/
X.400: Original encoded information types:
X.400:   Basic information type = A000
X.400:     1... .. = Undefined
X.400:     .0.. .. = No tLX
X.400:     ..1. .... = IASText
X.400:     ...0 ..... = No g3Fax
X.400:     ....0..... = No tIF0
X.400:     .....0.... = No tTX
X.400:     .....0.... = No videotex
X.400:     .....0.... = No voice
X.400:     .....0.... = No mPD
X.400:     .....0.... = No tIF1
X.400: Content type = 2 (P2)
X.400: UA content id = 880726 14:53:53
X.400: Priority = 2 (Urgent)
X.400: Per message flag = C0
X.400:   1... .. = Disclose recipients
X.400:   .1.. .. = Conversion prohibited
X.400:   ..0. .... = No alternate recipient allowed
X.400:   ...0 .... = No content return request
X.400: Recipient info:
X.400:   Recipient: /C=US/ADMD=ATTMAIL/PRMD=retix/0=sun/PN=donoghue.barbara/
X.400:   Extension identifier = 1
X.400:   Per recipient flag = A0
X.400:     1... .. = responsibility flag on
X.400:     .01. .... = basic report request
X.400:     ...0 1... = basic user report request
X.400: Recipient info:
X.400:   Recipient: /C=US/ADMD=ATTMAIL/PRMD=retix/0=sun/PN=seifrid.elena/
X.400:   Extension identifier = 2
X.400:   Per recipient flag = D0
X.400:     1... .. = responsibility flag on
X.400:     .10. .... = confirmed report request
X.400:     ...1 0... = confirmed user report request
X.400: Trace information:
X.400:   Global domain identifier: /C=US/ADMD=ATTMAIL/PRMD=RETIX/
X.400:   Arrival = 26 Jul 1988 14:53:58-0700
X.400:   Action = 0 (Relayed)
X.400: Internal trace info:
X.400:   MTA name = UAX
X.400:   Arrival = 26 Jul 1988 14:53:58-0700
X.400:   Action = 0 (Relayed)
X.400:
X.400: ----- X.400 Interpersonal Messaging Protocol (P2) -----
X.400:
X.400: UAPDU type = Interpersonal Message (length = indefinite)
X.400: Heading:
X.400:   IP message id: /C=US/ADMD=ATTMAIL/PRMD=RETIX/0=UAX/PN=SEIFRID.ELENA.
X.400:   880726 14:53:53
X.400:   Originator: /C=US/ADMD=ATTMAIL/PRMD=retix/0=sun/PN=Seifrid.Elena.D/
X.400:   Primary recipient:
X.400:     O/R Descriptor: /C=US/ADMD=ATTMAIL/PRMD=retix/0=sun/PN=donoghue.bar
X.400:     Reply request = FALSE
X.400:   Primary recipient:
X.400:     O/R Descriptor: /C=US/ADMD=ATTMAIL/PRMD=retix/0=sun/PN=seifrid.elen
X.400:   Subject = Test message on Retix X.400 network
X.400: Body:
X.400:   IASText = "Testing for Andre...."
X.400:   Unidentified [1720 bytes]
X.400:

```

Frame 9 of 171

Use TAB to select windows

1 Help 2 Set mark

4 Zoom out

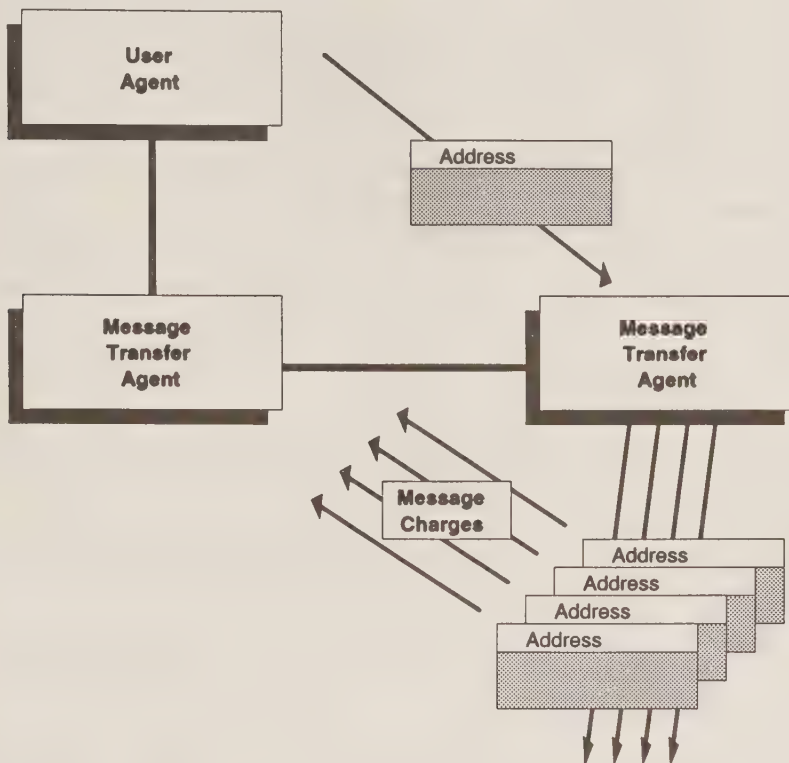
5 Menus

6 Display options

7 Prev frame

8 Next frame

10 New capture



13-25 Expansion of Distribution Lists

of the distribution lists used to get it to this point. The expander checks the list to detect any recursion.

Notifications regarding a distribution list can come from the distribution list expansion point (e.g., submit permission denied) or at the ultimate destination (message undeliverable).

A distribution list is a special form of O/R address. The message is handled just as any other name is until it reaches the MTA responsible for that O/R address.

## Message Stores

Message stores are useful for low-availability workstations like a PC. Since the PC can't always accept its message, there needs to be a way of storing the message until the PC is done with one task and turns to mail. Message stores are also useful for a user that simply has a terminal and no computer.

Figure 13-26 shows the elements of service provided by a message store. As can be seen, the service is pretty simple. The user can store, delete,



<b>Basic Elements</b>	Stored message deletion
	Stored message fetching
	Stored message listing
	Stored message summary
<b>Additional Elements</b>	Stored message alert
	Stored message auto-forward
Source: CCITT Blue Book VIII.7 - X.400 (p. 37)	

### 13-26 Message Store Elements

fetch, and list messages. Additionally, alerts can be sent by the message store and auto-forwarding can be performed.

The message store is thus just a program that accepts messages on behalf of a user, getting them out of the message transfer system.

## Gateways

Gateways allow the interconnection of different message handling systems. Telex users, for example, can access X.400 users through the use of the telex access units. Perhaps the most interesting of the gateways is the postal delivery service interconnection.

Obviously, this type of gateway is one way: it would be nearly impossible to take pieces of paper in unknown formats and turn them automatically into X.400 mail messages. The reverse, however, is almost trivial. All a gateway has to do is take an incoming message, print out the body part on a piece of paper, put the postal address on the top, and somehow get the postal service to pick it up and deliver it.

The advantage of this type of gateway to the user is enormous: a single user interface can be used for interacting with all message recipients.

While the author has not seen an X.400 postal delivery system access unit implemented, MCI Mail does have a similar service. When sending a message (or composing a distribution list), the MCI Mail user simply types in a name. If the name is not a valid MCI Mail address, the system comes back and asks the name of the delivery system. Valid options include X.400, telex, fax, and postal delivery.

For postal delivery, the system asks the user for a valid address. The message body and an envelope are printed, and the message is simply dispatched into the maw of the postal service. Charges are paid for by MCI Mail and then recovered on the user's monthly bill.

Figure 13-27 shows the basic elements of service for a postal delivery gateway. Obviously, the access unit must be capable of providing a basic



<b>Basic Elements</b>	Basic physical rendition
	Ordinary mail
	Physical forwarding allowed
	Undeliverable mail with return of physical message
<b>Essential Elements</b>	Counter collection
	Express mail service
	Special delivery
<b>Additional Elements</b>	Additional physical rendition
	Counter collection with advice
	Delivery via Bureau fax service
	Physical delivery notification by MHS
	Physical delivery notification by PDS
	Notification forwarding prohibited
	Registered mail
	Registered mail to addressee in person
Request for forwarding address	
Source: CCITT Blue Book VIII.7 - X.400 (p. 36)	

### 13-27 Postal Unit Elements

physical rendition of the message and must allow the postal system to perform its normal tasks such as forwarding of the mail.

The user should be able to pick special options such as counter collection, express mail service, or even special delivery. Remember that the access unit provider is paying for these services and will then charge back the X.400 user.

A variety of additional elements, such as registered mail or notification of a forwarding address, require a little bit closer coordination with the postal service. The basic elements can easily be provided by a corner service bureau. The additional elements may require that the service be provided by, or at least approved by, the postal service.

### Munging to X.400

Connecting the Internet world, based on DNS, SMTP, and RFC 822, to the X.400 world is quite a challenge. In the Sun world, connecting to X.400 is done as an access unit, the SunLink MHS software. This is simply a gateway between the X.400 world and the native SMTP or UUCP-based environments.

Two very complex RFCs discuss how the two worlds can be linked together via a gateway. Going from the Internet world into X.400 is not terri-

From ICS.UCI.EDU!na-mhsnews-request Wed Oct 24 15:43:24 1990 remote from uunet  
Received: by malamud.com (UUL1.3#5653)  
from uunet with UUCP; Wed, 24 Oct 90 14:37:58 PST  
Received: from ics.uci.edu by uunet.uu.net (5.61/1.14) with SMTP  
Id AA18872; Wed, 24 Oct 90 15:43:24 -0400  
Received: from ics.uci.edu by ICS.UCI.EDU id aa08180; 24 Oct 90 10:21 PDT  
Received: from ics.uci.edu by ICS.UCI.EDU id aa08167; 24 Oct 90 10:19 PDT  
Received: from kwai.inria.fr by ICS.UCI.EDU id aa08102; 24 Oct 90 10:19 PDT  
X400-Received: by /PRMD=inria/ADMD=atlas/C=FR/;  
Relayed; 24 Oct 90 18:20:38+0100  
X400-Received: by /PRMD=emse/ADMD=atlas/C=fr/;  
Relayed; 24 Oct 90 18:14:48+0100  
Date: 24 Oct 90 18:14:48+0100  
From: Paul-Andre Pays <uunet!mars.emse.fr!pays>  
Message-Id: <9010241714.AA00412@mars.emse.fr.emse.fr>  
To: mhsnews@ICS.UCI.EDU, mws@sitman.ssw.COM  
Subject: re: your mail

—— Begin Included Message ——

X400-Received: by /PRMD=emse/ADMD=atlas/C=fr/;  
Relayed; 24 Oct 90 18:02:05+0100  
Date: 24 Oct 90 18:02:05+0100  
From: Mark Sitler <mws@sitman.ssw.com>  
Message-ID: <9010241502.AA15753@sitman.ssw.com>  
To: mhsnews@ics.uci.edu  
Autoforwarded: True

Status: RO

### 13-28 A Message Crossing Three MHSs

bly complicated. There needs to be a way to handle address translation across the gateway. Aside from that, most of the simple header items in RFC 822 map quite easily into the X.400 world.

The tough part is moving from X.400 to RFC 822. X.400 provides significantly more functionality. Among other problems, X.400 provides for more than just ASCII body parts, and there can be several body parts in a single message.

The general rule is that text converts quite easily. For body parts that can't convert, the gateway can either reject the message or convert as much as possible, inserting a "Gateway Ate This Part" indicator where it fails.

Conversion of other items is necessary as well. Time, for example, is represented differently in the two systems. X.400 has provisions for integers, which must be translated into some ASCII representation. All these details need to be converted.

Another difficulty is the question of header items. In the X.400 world, many header items require work to be done: receipts sent, conversions performed, automatic forwarding sent. Very few of these are present in the RFC 822 world.

Figure 13-28 shows one one final illustration, a message that crossed over from X.400 to SMTPland, and finally to UUCPland. As the reader can see, the interconnection of messaging environments is certainly a reality. For more information on exactly how the protocols work, the reader should consult Marshall Rose's *The Little Black Book*.

## For Further Reading

David H. Crocker, *Standard for the Format of ARPA Internet Text Messages*, RFC 822, August, 1982.

Frey, Donalyn, and Adams, Rick, *!%@:: A Directory of Electronic Mail* (2nd ed.). O'Reilly & Associates (Sebastopol, CA: 1990).

S. Kille, *Mapping Between X.400(1988)/ISO 10021 and RFC 822*, RFC 1148, March, 1990.

———, *Mapping between X.400 and RFC 822*, RFC 987, June, 1986.

LaQuey, Tracey L., ed., *The User's Directory of Computer Networks*. Digital Press (Maynard, Mass: 1990).

Tim O'Reilly and Grace Todino, *Managing UUCP and Usenet*, O'Reilly and Associates (Newton, Mass: 1988).

Craig Partridge, *Mail Routing and the Domain System*, RFC 974, January, 1986.

J. Postel, *Simple Mail Transfer Protocol*, RFC 821, August, 1982.

Quarterman, John S., *The Matrix*. Digital Press (Maynard, Mass: 1990).

Rose, Marshall, *The Little Black Book: Mail Bonding with OSI Directory Services*. Englewood Cliffs, N.J.: Prentice Hall, 1991.

Marshall T. Rose, Einar A. Stefferud, *Proposed Standard for Message Encapsulation*, RFC 934, January 1985.

Sun Microsystems, *SunNet MHS System Administration Manual*, Part No: 800-5009-1300, Version 50 of 05 October 1990.

Grace Todino, *Using UUCP and Usenet*, O'Reilly and Associates (Newton, Mass: 1987).





# Files, Printers, and Terminals



# Files, Printers, and Terminals

## Overview

This chapter, along with Chapter 12, is a grab bag. Here applications commonly found on the Internet are explained. This includes the two basic services, the virtual terminal service (Telnet) and the File Transfer Protocol (FTP). In addition to FTP, an Internet application, the File Transfer Access and Management (FTAM) protocols defined in OSI are included. Sun computers are increasingly being used in dual TCP/OSI environments. The SunLink OSI gateway software provides access to FTAM and other OSI-based services.

We start with Telnet, then move on to FTP and FTAM. After the two file transfer protocols, we will move on to the grab bag within the grab bag. First, two resource location protocols, *whois* and *finger*, are explained. Next, the line printer daemon (LPD) protocol for remote printing. Finally, the trivial file transfer protocol, used for booting diskless nodes as well as a variety of other tasks is discussed.

## Telnet

Telnet is a TCP-based service that allows a user to send data interspersed with control information. The typical use is a remote login, but the service is general and can be used for a variety of applications. There are three basic concepts behind Telnet:

- The network virtual terminal
- Negotiated options
- A symmetric view of terminals

The network virtual terminal defines an abstract device on which both the client and the server can perform operations. The user types and the information is turned by the Telnet client process into an operation on the network virtual terminal. Why perform such an abstraction? There are so many differences among different terminals in the Unix world that managing terminals is an extremely device-dependent operation. Agreeing on a common set of virtual operations allows local implementors to map their device-dependent operation into the agreed-upon set.

Negotiated options means that the Telnet protocol starts very simply but can get more complicated. If the client and the server Telnet both agree, bells and whistles (literally) can be added as special control characters, leading to a more sophisticated network virtual terminal. If they don't agree, the network virtual terminal is a very simple terminal.

Symmetry means that either side of a connection can propose an option to be used. This can easily lead to loops, so a few rules are imposed. For example, a Telnet process can only send an option request to change something, not simply as an announcement of the current mode (otherwise the announcement triggers another announcement, which can cause a loop).

A second rule to prevent loops is that if a node gets a request to enter a mode it is already in, the node should not answer. Nonresponse prevents endless loops of requests. A request to change modes must be answered, even if the node simply refuses to make the change. Both these rules, in addition to preventing loops, also prevent "option storms" or at least limit them to the beginning of the session after which one hopes they gradually subside.

Options are based on four basic verbs:

- Will: I desire to start this service.
- Won't: I will not start this service.
- Do: You should start this service.
- Don't: You should not start this service.

One Telnet process might send a *will option* command, indicating it wants to do something. If the response is *do option*, the negotiation is successful. If the answer is *don't option*, the negotiation is unsuccessful. Likewise, to ask the remote node to do something, the node would send *do option* and the answer would be either *will option* or *won't option*.

This structure allows nodes to refuse options that they don't know about. Once the exchange has taken place, there can be further negotiation for suboptions, but only if both nodes agree to support the service.

### *The Network Virtual Terminal*

The network virtual terminal (NVT) is a bidirectional character device with a printer and a keyboard. The printer responds to incoming data the key-



Command	Code	Meaning
SE	240	End of subnegotiation parameters.
nop	241	No operation.
data mark	242	Data stream portion of a sync (should also set the TCP urgent pointer).
break	243	Network virtual terminal break character.
ip	244	Interrupt process.
ao	245	Abort output.
ayt	246	Are you there?
ec	247	Erase character.
el	248	Erase line.
ga	249	Go ahead.
sb	250	Subnegotiation begin.
will	251	Negotiation commands, followed by option code.
wont	252	
do	253	
dont	254	
iac	255	Interpret as command.

#### 14-1 Telnet Command Characters

board produces outgoing data. Note that outgoing data may also end up back at the printer if echoing is desired. Echoes can be local or remote.

The underlying transport service is intrinsically full duplex, but the default NVT is a half-duplex device operating in a line-buffered mode (it saves a line locally and then sends it all off at once). Options allow this to change to a character-buffered device (for full-screen editing) or a full-duplex device.

The basic data stream is 7-bit ASCII characters, limiting the range to the lower half of the ASCII character set. In addition, there are a series of control characters interspersed with the data (see Fig. 14-1). All the commands begin with the special Interpret As Command (IAC) control character, followed by one or more bytes of control information.

A special signal provided by Telnet is the sync signal. Sync is a TCP urgent notification, coupled with the Telnet data mark (DM). Urgent notification is not subject to TCP flow control. When the remote node gets the urgent signal, it immediately scans the incoming data stream for interesting data: an abort control signal, for example.

The data mark indicates the end of the interesting area. Normally, this signals the remote end to resume normal processing. For example, a user might send a command to a remote operating system. Realizing that the

command could take 48 hours to execute, the user hits a local abort signal, say control/C.

The Telnet module sees the control/C and realizes that this is a special character. Rather than just send the character off and wait the 48 hours for the remote host to finish the first command (and then finally take a look at the abort signal), Telnet will send a TCP urgent notification off. The remote Telnet will see that and let the remote program (in this case a command processor) know that an interrupt occurred. It would then hand all remaining data up to the data mark (a control/C) to the remote operating system, which would presumably abort the command. The data mark would indicate the end of the interesting data and the user would continue with the next command.

### *Subnegotiation and Options*

A wide variety of different options are defined in a variety of RFCs (see Fig. 14-2). Some are in the basic Telnet RFC, but many others were defined later as the hardware and software associated with terminal sessions increased.

Subnegotiation protocols are defined for several of the options. Both sides must first agree that the option is to be done, and then automatically enter the subnegotiation. Subnegotiations are passed using the SB (250) and SE (240) bytes (each with an IAC right before them). An example of a subnegotiation is extending the options list. There are 254 options defined, but an additional 256 can be added using the extended options list (EXOPL) code:

```
IAC SB EXOPL DO option IAC SE
```

This command starts with an IAC to indicate that a command follows. Next, a subnegotiation begin (SB) indicates that a particular subnegotiation will follow, in this case the extended options list (EXOPL). The DO command indicates that the node would like to perform an option. At the end is the IAC subnegotiation end (SE) to indicate that normal data processing can resume.

If that extended option in turn needs further negotiation, we might see something like this:

```
IAC SB EXOPL SB OPTION PARAM SE IAC SE
```

Again, the subnegotiation is signalled and the EXOPL listed. This is followed by a nested subnegotiation which includes an option and a parameter. The first subnegotiation ends, followed by the outer subnegotiation. A bit confusing to the eye, but easy for a program to parse.

Document	Title
RFC 1116	Telnet Linemode Option
RFC 1097	Telnet Subliminal-Message Option
RFC 1096	Telnet X Display Location Option
RFC 1091	Telnet Terminal-Type Option
RFC 1080	Telnet Remote Flow Control Option
RFC 1079	Telnet Terminal Speed Option
RFC 1073	Telnet Window Size Option
RFC 1053	Telnet X.3 PAD Option
RFC 1043	Telnet Data Entry Terminal Option: DODIIS Implementation
RFC 1041	Telnet 3270 Regime Option
RFC 946	Telnet Terminal Location Number Option
RFC 933	Output Marking Telnet Option
RFC 930	Telnet Terminal Type Option
RFC 927	TACACS User Identification Telnet Option
RFC 885	Telnet End of Record Option
RFC 884	Telnet Terminal Type Option
RFC 861	Telnet Extended Options List Option
RFC 860	Telnet Timing Mark Option
RFC 859	Telnet Status Option
RFC 858	Telnet Suppress Go Ahead Option
RFC 857	Telnet Echo Option
RFC 856	Telnet Binary Transmission
RFC 855	Telnet Option Specifications
RFC 854	Telnet Protocol Specification

## 14-2 Telnet Options

As a default, the Sun Telnet implementation asks for remote echo, suppress go ahead, and terminal type options. In addition, it can handle echo, binary, suppress go ahead, set terminal type, and timing options.

### *Output Discarding*

Many operating systems have something like a control/O: discard all further output. Many users on a terminal connected via Telnet protocol will expect similar behavior from the remote system as it gets with the local hardwired system.

The problem is that by the time the abort output character gets to the host operating system, there may be thousands of characters already sent by the Telnet server but not displayed on the user's terminal. The client



system, not knowing what character is used on the remote host to discard output, simply passes the control/O through as normal data.

The output discarding option is a way for the client system to know that a special output discard signal was typed. The Telnet client needs to know which remote operating system specific command characters are assigned to the abstract functions of abort output or interrupt process.

A potential problem here is that the Telnet data stream has embedded commands, including option negotiations. This means that a node cannot discard all data indiscriminantly—only “normal” data should be dumped. One method for handling this is to let the remote host find the discard output character. As soon as the host operating system gives a discard signal, the Telnet server must quickly notify the Telnet client that a discard is in progress. The notification is put on the very next TCP segment sent to the client (even if there are many bytes of TCP data buffered at the server).

A better way to handle this problem is using the urgent bit in TCP packets. The TCP urgent bit identifies an interesting byte in the data stream. In the interesting byte is the two-character sequence IAC DM. The IAC specifies that a control character appears in the data stream. The data mark (DM) is a Telnet-inserted mark that acts as a synchronization signal. The hosts know they can discard all information up to the DM.

When a Telnet client system recognizes the operating system-specific command character assigned to a function, the client then translates this operating system-specific character into the standard Telnet representation:

- IAC AO (abort output)
- IAC IP (interrupt process)

When client detects the operating system-specific character, it inserts the appropriate representation and puts it into the data stream, then puts a sync signal right after that. When the Telnet server gets the abort output, it then sends a sync back to the client. When the server gets an IP, it may send a sync signal.

### *Workstation Support*

Several Telnet options allow a workstation to make use of Telnet from within a windowing system. These options provide a mechanism to pass information about display window sizes and size changes from client to server. Negotiate About Window Size (NAWS), documented in RFC 1075, allows window dimensions of 65536 by 65536.

Another option, RFC 1091, extends the existing Telnet Terminal Type (TT-TYPE) to allow the server to review the client’s list of supported terminal emulations and select the one it prefers. Later (e.g., when the application asks for a change) the server can pick a new type.



RFC 1096 allows a node to pass an X Windows System display location string across a Telnet connection without explicit user intervention. A new option, X-Display-Location (XDISPLOC), is specified that allows Telnet clients and servers to negotiate. If the client and server agree to negotiate the option, the server sends a Telnet subnegotiation sequence to query for an ASCII string in the format `host:display_number[.screen_number]`.

### *Telnet Traffic*

Figures 14-3 through 14-6 show some typical Telnet traffic. Figure 14-3 shows some several frames of traffic between two nodes. Notice that the traffic begins to be echoed back, one character at a time, one packet per character. Figure 14-4 shows a Telnet session over TCP. Notice the packet acknowledgments.

Figures 14-5 and 14-6 show an option negotiation. The first node lays several items on the table, such as a desire to perform echoing of traffic and to set the terminal type. Figure 14-6 shows that the option Send Location was refused.

### The File Transfer Protocol

The File Transfer Protocol (FTP) is one of the big three TCP/IP applications, along with Mail and Telnet. FTP dates back to 1971 with RFC 114. While a bit frayed around the edges with time, FTP continues to serve a large role in the Internet. There are many other data access mechanisms, notably the Network File System and FTAM. The Network File System is a much more integrated type of access mechanism and is given a lengthy treatment in its own chapter.

FTAM, discussed after FTP, provides a different service than FTP. Based on OSI, the FTAM service provides file access: records, attributes, and other components of files. FTP, by contrast, has a more limited scope: the transfer of entire files.

FTP has a significant advantage over FTAM: it is more broadly implemented. As a standard part of the TCP/IP suite, it is available on every major operating system, often in public domain versions. Though NFS has largely replaced FTP within a workgroup or enterprise network, it is still used extensively in internetworking situations which security and administrative boundaries that limit NFS access.

FTP is based on two TCP connections. The first is a Telnet connection and is used for sending commands and responses (see Fig. 14-7). The second connection is used for the actual exchange of the data.

The architecture itself is based on two components: the protocol interpreter and the data transfer process. The protocol interpreter (PI) is a Tel-

Delta T	DST	SRC	
1.2167	Intrln001789+DECnet000404	Telnet C PORT=3460	s
0.1292	Intrln001789+DECnet000404	Telnet C PORT=3460	o
0.2198	Intrln001789+DECnet000404	Telnet C PORT=3460	n
0.1199	Intrln001789+DECnet000404	Telnet C PORT=3460	o
0.2400	Intrln001789+DECnet000404	Telnet C PORT=3460	x
0.8296	Intrln001789+DECnet000404	Telnet C PORT=3460	<0D><0A>
3.6307	DECnet000404+Intrln001789	Telnet R PORT=3460	S
0.0188	DECnet000404+Intrln001789	Telnet R PORT=3460	O
0.0235	DECnet000404+Intrln001789	Telnet R PORT=3460	N

DETAIL

Telnet:----- Telnet data -----  
 Telnet:  
 Telnet:s  
 Telnet:

Frame 9 of 40

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 14-3 Remote Echo

SUMMARY	Delta T	DST	SRC	
M 1		Bridge00AC94+0000D0000070	Telnet C PORT=25860	<1B>[A
2	0.0087	0000D0000070+Bridge00AC94	TCP D=25860 S=23	ACK=5258582
3	0.0640	0000D0000070+Bridge00AC94	Telnet R PORT=25860	<02><0D><1B>
4	0.0127	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252802
5	1.0142	Bridge00AC94+0000D0000070	Telnet C PORT=25860	<0D>
6	0.0089	0000D0000070+Bridge00AC94	TCP D=25860 S=23	ACK=5258582
7	0.0296	0000D0000070+Bridge00AC94	Telnet R PORT=25860	<0D><0A>
8	0.0108	Bridge00AC94+0000D0000070	TCP D=23 S=25860	ACK=5252802
9	1.2534	0000D0000070+Bridge00AC94	Telnet R PORT=25860	<0D>; Union

DETAIL

TCP:----- TCP header -----  
 TCP:  
 TCP: Source port = 23 (Telnet)  
 TCP: Destination port = 25860  
 TCP: Sequence number = 52528008  
 TCP: Acknowledgment number = 52585820  
 TCP: Data offset = 20 bytes  
 TCP: Flags = 10  
 TCP: ..0. .... = (No urgent pointer)

Frame 2 of 1354

Use TAB to select windows

1 Help 2 Set mark 4 Zoom in 5 Menus 6 Display options 7 Prev frame 8 Next frame 10 New capture

## 14-4 Telnet over TCP

SUMMARY	Delta T	DST	SRC	
M 1		KinetxD005F1+DECnet000E60	Telnet R PORT=10856 <0D><0A>vulc	
2	0.1096	DECnet000E60+KinetxD005F1	TCP D=23 S=10856 ACK=2786525	
3	0.1399	Encore00235B+Exceln402165	Telnet R PORT=781 <0A> Job Line	
4	0.1176	Exceln402165+Encore00235B	TCP D=23 S=781 ACK=33561916	
5	0.0051	Encore00235B+Exceln402165	Telnet R PORT=781 Origin<	
6	0.0511	KinetxD005F1+DECnet000E60	Telnet R PORT=10856 IAC Will Ech	
7	0.0568	DECnet000E60+KinetxD005F1	Telnet C PORT=10856 IAC Will Sup	
8	0.0107	KinetxD005F1+DECnet000E60	Telnet R PORT=10856 <07><0D><0A>	
9	0.0076	KinetxD005F1+DECnet000E60	TCP D=10856 S=23 ACK=1115750	

## DETAIL

```

Telnet:----- Telnet data -----
Telnet:
Telnet:IAC Will Echo
Telnet:IAC Will Suppress go-ahead
Telnet:IAC Do Suppress go-ahead
Telnet:IAC Do Send location
Telnet:IAC Do Terminal type
Telnet:IAC Do Terminal location number

```

Frame 6 of 1007

Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

## 14-5 Telnet Option Negotiation

SUMMARY	Delta T	DST	SRC	
M 1		KinetxD005F1+DECnet000E60	Telnet R PORT=10856 <0D><0A>vulc	
2	0.1096	DECnet000E60+KinetxD005F1	TCP D=23 S=10856 ACK=2786525	
3	0.1399	Encore00235B+Exceln402165	Telnet R PORT=781 <0A> Job Line	
4	0.1176	Exceln402165+Encore00235B	TCP D=23 S=781 ACK=33561916	
5	0.0051	Encore00235B+Exceln402165	Telnet R PORT=781 Origin<	
6	0.0511	KinetxD005F1+DECnet000E60	Telnet R PORT=10856 IAC Will Ech	
7	0.0568	DECnet000E60+KinetxD005F1	Telnet C PORT=10856 IAC Will Sup	
8	0.0107	KinetxD005F1+DECnet000E60	Telnet R PORT=10856 <07><0D><0A>	
9	0.0076	KinetxD005F1+DECnet000E60	TCP D=10856 S=23 ACK=1115750	

## DETAIL

```

Telnet:----- Telnet data -----
Telnet:
Telnet:IAC Will Suppress go-ahead
Telnet:IAC Won't Send location
Telnet:IAC Will Terminal type
Telnet:IAC Won't Terminal location number

```

Frame 7 of 1007

Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

## 14-6 Response to Negotiation



Access Control Commands	
User	Whatever required by some system. Could be just username, but might also need password and/or account information. Some servers allow a new user command to be entered at any point.
Pass	Password. Must be immediately preceded by username command. User FTP should mask this on the user's screen.
ACCT	Account information. There is a way to automate this. If the response to the PASS command is 332, account information is required. If reply is 230, account information is not needed. If it will be needed later, a 332 or 532 response is returned.
CWD	Change Working Directory.
CDUP	Change to Parent Directory. Special case of CWD.
SMNT	Structure Mount. Lets a user mount a different file system data structure.
REIN	Reinitialize. Terminates a user, flushes all I/O and account information (but allows transfer in progress to complete). All parameters to default settings, control connection left open.
QUIT	Logout. Closes control connection if file transfer not in progress. If in progress, remains open for result response. To continue the session with another user, use REIN instead of quit.
Transfer Parameter Commands	
PORT	Data Port. Default is source port on user, but if used, consists of a 32-bit IP address plus a 16 bit TCP port number. This is broken up into 8-bit fields and sent as a decimal number, separated by commas.
PASV	Passive. Requests the server DTP to listen on a data port that is not the default and wait for a connection instead of initiating one upon receipt of transfer command. The result of this is the host and port address that the server listens on.
TYPE	Representation Type. Some types take a second parameter. First parameter is single character, second is single character or number (default ASCII nonprint).
	A (ASCII)
	C (ASA carriage control)
	E (EBCDIC)
	I (Image)
	N (nonprint)
	T (Telnet format effectors)
L	Local byte size.
STRU	File Structure (F default).
	F File.
	R Record.
	P Page.



MODE	Transfer Mode (S default).
	S Stream.
	B Block.
	C Compressed.
<b>FTP Service Commands (Arguments are usually a path name. )</b>	
RETR	Retrieve.
STOR	Store. Causes server DTP to accept data.
STOU	Store Unique. Perform the copy, but make sure that the resulting file name is unique.
APPE	Append (with create). Server accepts data and stores at server site.
ALLO	Allocate. Allows the user to allocate sufficient storage. Argument is decimal integer representing number of logical bytes to reserve. If the file is a record or page structure, the second argument is the maximum record/page size in logical bytes. Command should be followed by store or append command. The command is treated like a NOOP if the operating system does not need to reserve space for files.
REST	Restart. Argument is a server marker.
RNFR	Rename From. Immediately followed by RNTD command.
RNTD	Rename To.
ABOR	Abort. If current FTP service command already completed, reply will be 226 (abort successfully processed). If in progress, the reply will be 426 (service request terminated abnormally) followed by a 226.
DELE	Delete.
RMD	Remove Directory.
MKD	Make Directory.
PWD	Print Working Directory.
LIST	List files in directory (or information on a file if a filename is the argument). Null argument is user's current working or default directory. Data transfer is over the data connection.
NLST	Name List. Returns stream of file names, each separated by CRLF or NL. Used by programs (i.e., a multiple get function).
SITE	Site Parameters. Site specific. Find out using help site command.
SYST	System. Returns the name of the operating system.
STAT	Status. Sends status command (possibly during the middle of a file transfer). During transfer, returns the status of that transfer. Otherwise, the argument is a file name and file status information is returned over the control connection.
HELP	Can accept a parameter.
NOOP	Does nothing.

#### 14-7 (Cont.) FTP Commands

net-speaking process used to exchange commands. It adds value to Telnet by defining what actions are accomplished by each of the commands.

The data transfer process (DTP) handles the movement of the data between machines. The DTP doesn't actually worry about the remote DTP module: it operates on a network virtual file system, which is an abstraction of the concept of a file. What this means is that data is represented in a machine-independent format.

One interesting artifact of the separation of the protocol interpreter from the data channel is that it is possible for a user to transfer files between two other hosts. Control connections are set up to two different servers and then a command is issued to set up a data connection between the two servers. Keeping the control connections open allows progress to be monitored and subsequent commands issued.

### *Data Representation*

Since different machines have different architectures, FTP makes a few rudimentary provisions for data independence. Byte size is always 8 bits, for example. A system may store bytes any way it wants, but it uses an 8-bit representation on the network.

ASCII (or EBCDIC) codes are used for representation of text. Again, even ASCII varies among systems. A subset of ASCII, known as netASCII is used. This is a 7-bit ASCII (printable characters plus a few control characters) in which the most significant bit is always zero.

FTP handles data representation by allowing a user to specify a representation type. Type can be implicit (ASCII/EBCDIC) or explicit by defining a byte size known as the logical byte size. The logical byte size is different from the transfer byte size. The transfer byte size is always 8 bits: data is grouped in octets of 8 bits each. However, several of these transfer bytes may be used to make a large logical byte size. If the logical byte size is 14 (for some strange architecture), it would take two transfer bytes to represent a single logical byte, with two bits wasted.

In addition to specifying ASCII and EBCDIC, these two types can include a second parameter which adds format control information to the stream of characters.

Two major types of format control are Telnet style and Fortran Carriage Control (also known as ASA). Telnet style recognizes vertical movement characters such as the carriage return, line feed, new line, vertical tab, and form feed. If data is going right to the screen, or if a system needs to pick out the special characters to translate them into internal representation, this typing makes recognition easier. With ASA format control, the first character of a line indicates how vertical format control is to be done for that line. This is often used with old line printers.

There are four basic characters that can be present on the first column of a line:

- Blank means move paper up one line.
- 0 means move up two lines.
- 1 means move up to top of page.
- + means no movement (overprint).

FTP allows a data structure to be named for the transfer. Three basic data structures have been defined:

- File structure (no internal structure: contiguous data bytes)
- Record structure
- Page structure (independent indexed pages)

A file structure is the default and means that the data is simply a string of contiguous bytes with no structure. If the structure command is not explicitly used, this is the default. The record structure says that the data is a series of records, each terminated with some end of record indicator such as a carriage return/line feed combination.

Page structure divides a file into a series of pages, each with a header. The header includes the length of the data in the page and the header. It also includes the page type and some option pages. The page type can be a normal page (i.e., data), a header page which has information for the file as a whole, or access controlled pages. Access controlled pages have an additional byte which limits access.

An interesting problem for FTP implementations occurs when correlating the FTP virtual file types into the internal method on the local file system. Unix systems, for example, are, by default, the file structure: a contiguous string of bytes. When the Unix system receives a record structure file, it must decide how to store that information internally to preserve the record structure of the file.

The data transfer procedure uses two TCP connections. This normally requires two ports on each end, one for each connection. A user may specify ports explicitly, but there are defaults. On the client side, the default data port is the same as the connection control port. On the server, the data port is the port adjacent (minus 1) to the control connection port.

Instead of a default port, the user can specify any address needed. For example, the passive data transfer port (the one on the client side usually) could be on a second server. The command would say to take a file from the first server to the second server.

Actual transfer of the data can be in any one of three modes. Normally, stream mode is used. This passes the data through with little or no processing. In this mode an end of file (EOF) mark is added at the end of the data



stream by the sending host, effectively closing the data connection. In stream mode, the connection has to be reopened for other files.

The other two transmission modes are block mode and compressed mode. In block mode, the data is formatted as a series of data blocks preceded by a header. The header contains a count field and a descriptor code. The descriptor identifies things like the last block in a file, the last block in a record, suspect data, or a restart marker.

The suspect data indicator is not really a recovery mechanism but is used to mark suspect data, as in the case of a read error from a magnetic tape drive. The restart markers are used for error recovery. Note that this is not a question of lost bits or scrambled data: these issues are handled by the underlying TCP service. Instead, here the concern is about a failure in the host, the FTP process, or the underlying network (e.g., a gateway fails).

The restart marker flag in the header indicates that this block has a restart marker. The marker is something that has meaning to the sender of the data. It could be a bit count, a record count, or even the label for a magnetic tape. The receiving FTP process notes the position of the marker and gives this information to the user. If there is a system failure, the user restarts the transfer by identifying the last marker point.

Compressed mode is the same as block mode with the additional function of (oddly enough) data compression. Compression takes multiple copies of a single character (e.g., a space) and compresses them to a byte count plus the character. If there are no repetitions in the data, there is no compression. For repetitive data, two kinds can be identified. White space is the most common type. Repetitive white space is represented in compressed mode as a single byte. The first 2 bits of the byte are 1, followed by a 6-bit byte count. Up to 64 spaces can be represented as a single byte of data.

How is a regular byte distinguished from compressed white space? In the netASCII character set, the first bit is always zero, since netASCII is limited to the basic characters. If bit 1 is set to 0, this must be a character. If two 1s are set, this is a white space count.

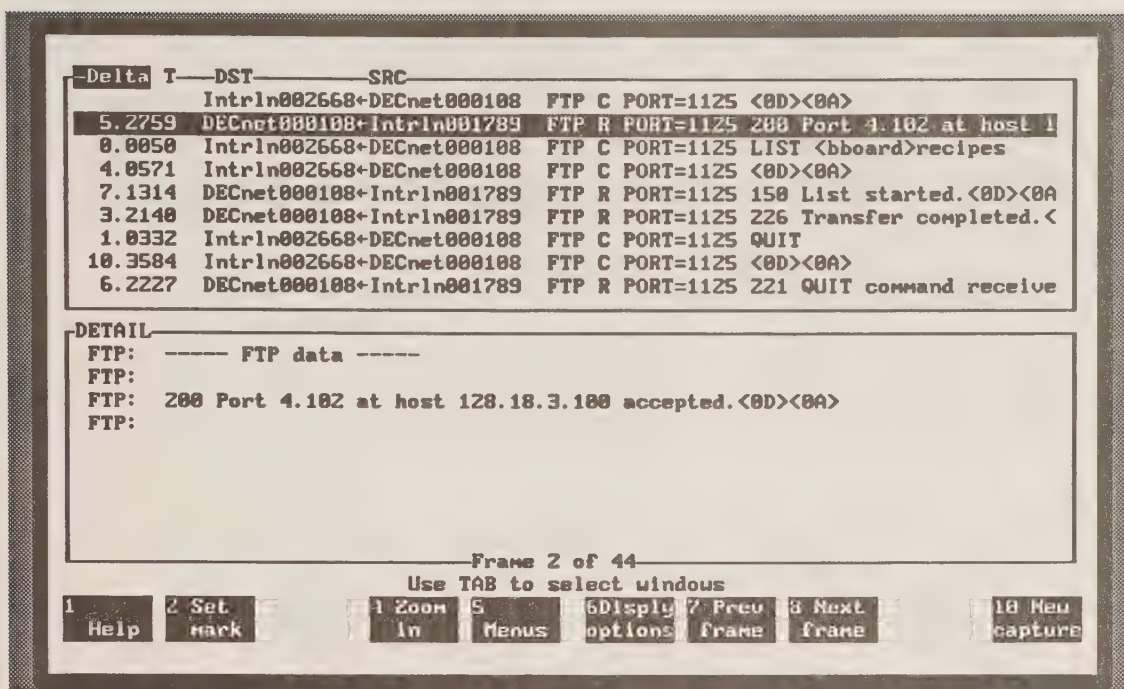
It is also possible to compress repetitions of other characters besides spaces. As the reader may have guessed, if the first 2 bits are set to 1 and then 0, this indicates a repetition of another character. These repetitions take a total of 2 bytes.

After the first 2 bits is a 6-bit byte count, indicating how many characters (up to 64) are repeated. The second byte of the escape sequence is the character that is being replicated.

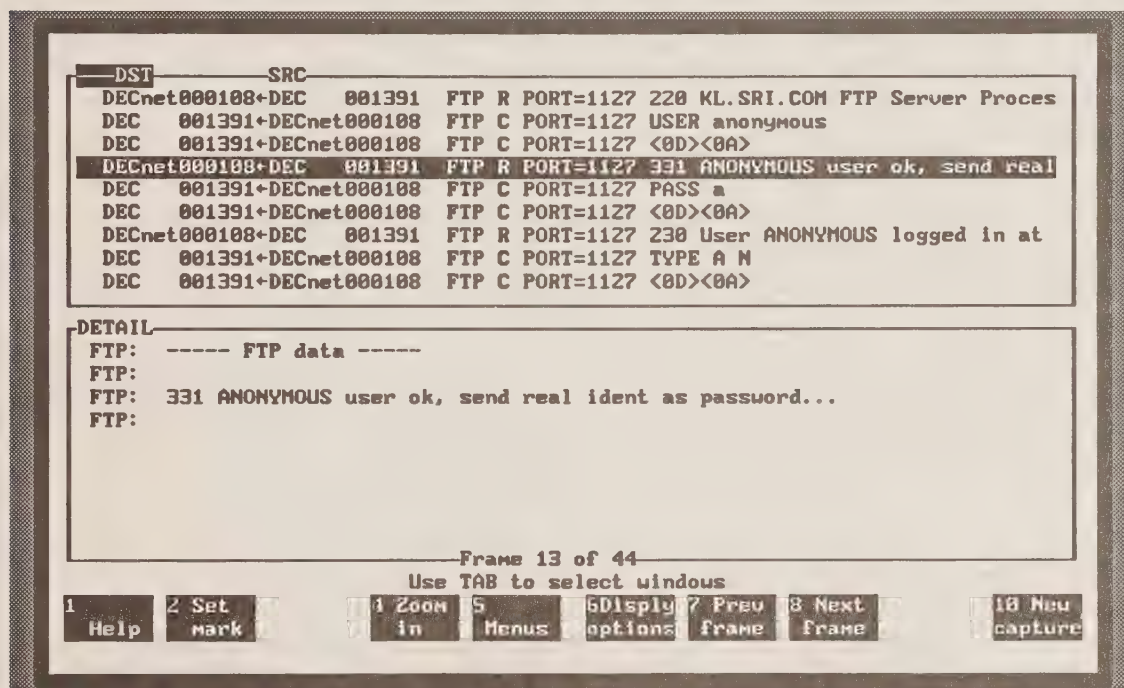
### *FTP Data*

Figures 14-8 through 14-10 show some typical FTP traffic on a network. In Figure 14-8, a host is accepting a session request. Notice the reply code as

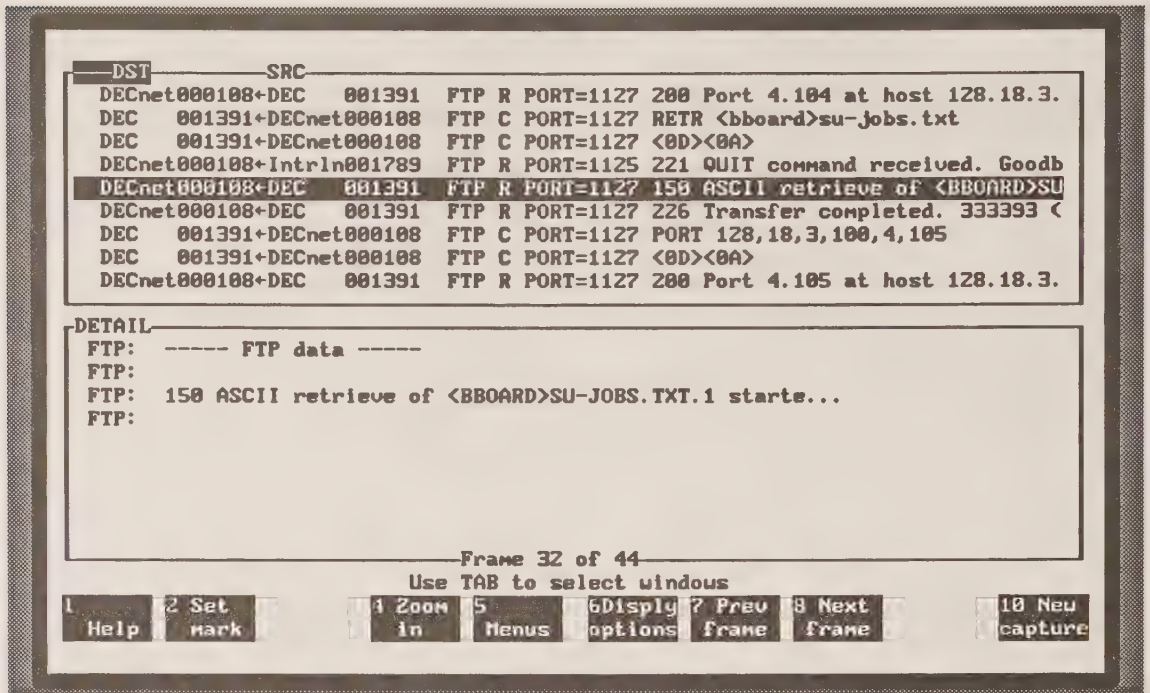




14-8 An FTP Session Initiation



14-9 Anonymous FTP



14-10 An ASCII Data Transfer

well as the reply text. The reply codes follow the same theory we saw in the discussion of the Simple Mail Transfer Protocol (SMTP). The session is quite short, consisting of a listing of the file <bboard>recipes.

Figure 14-9 shows a convention known as anonymous FTP. The convention says that if a user wants to access public files, she can do so without having a login on this particular host. Instead, the host allows any password for the username Anonymous (or sometimes the username FTP) and in return, the user is expected to transfer her real name in the password field.

Finally, Figure 14-10 shows an ASCII retrieve of a file. The user has sent the command RETR <bboard>su-jobs.txt. Notice just before the highlighted command there is a quit command. The quit command is for another FTP session occurring at the same time with the same DEC server.

## FTAM

The OSI File Transfer, Access, and Management (FTAM) protocols are similar in scope, although they have been designed with a variety of different file mechanisms in mind. As such, FTAM provides a general model, from which implementations will pick a specific subset. This subsetting of the protocol is quite important—two implementations will not necessarily inter-

connect at full functionality if they don't support the same subsets of the protocol or can't negotiate the same subset.

The general FTAM filestore model is of a single file, divided into a series of data units. A data unit consists of nodes structured in a tree. Attached to a node are zero or more data units. In a traditional file system, a node is a file and the data units are the data inside of the file. In FTAM, the model is based on a single file (there is no concept of multiple files inside a directory). In the FTAM model, a data unit is a record, page, block, or any other unit of structure inside a file.

We can model most file structures within this general model. A sequential text file, for example, consists of a single node under the root node, accompanied by a series of data units, one per record. A binary file consists of one node and one data unit.

Hierarchical file organizations, such as the Btree or the Indexed Sequential Access Method (ISAM), can be modeled as a multilevel tree. To lend some order to this potentially infinite tree, the FTAM model takes data units and structures them into file access data units (FADUs). A FADU is a portion of the tree down to the leaf nodes. Thus, the entire file can be considered a single FADU, just as lower-level portions can also be a FADU.

The FADU model of operation makes it conceptually easy to structure operations on the virtual file. Remember that actual operations on the file are performed by a local file operation. All that is needed by the FTAM protocols is an ability to communicate what operation needs to be done.

An FTAM session consists of a series of nested regimes. These regimes have certain operations that can be performed. An association regime, for example, is necessary before a file can be selected. Next, a file is opened, followed by any transfer operations. Notice that access is sequential: FTAM, like DAP, does not provide an asynchronous model of file processing.

The general FTAM model is broken up into functional modules, called service classes. Service classes structure the capabilities of the protocol into subsets. Service classes are levels of capabilities within the protocol and are negotiated by the two ends of a connection at connect time. FTAM has five service classes specified:

- File access allows access and manipulation at the record level.
- File transfer of whole files or parts of files.
- File management: reading and changing file attributes.
- File transfer and management includes creation and deletion of files.
- Unconstrained allows negotiation on the basis of available functional units instead of service classes.

Functional units are a more granular way of dividing up the capabilities of the protocol. Within a particular class of service, there are mandatory and optional functional units. You can always negotiate for optional func-



tional units within a class of service. The unconstrained service class is one with no mandatory functional units. The following are the ten basic functional units defined in FTAM:

- Kernel: basic file services for establishment of a connection
- Read
- Write
- File access: locating and manipulating FADUs within a file
- Limited management: file creation and deletion, reading attributes
- Enhanced file management: changing file attributes
- Grouping of operations so regimes can be efficiently established
- Recovery: re-creation of select or open regimes
- Restart data transfer
- FADU locking

Whereas the functional units and service classes deal with the services available in the protocol, document types limit what types of data may be contained in a virtual file being exchanged in an FTAM regime. Individual users may register their own file types with the proper authorities, specifying the structure of the data units and FADUs within the virtual filestore model.

In addition to user-defined document types, the ISO has defined five basic document types:

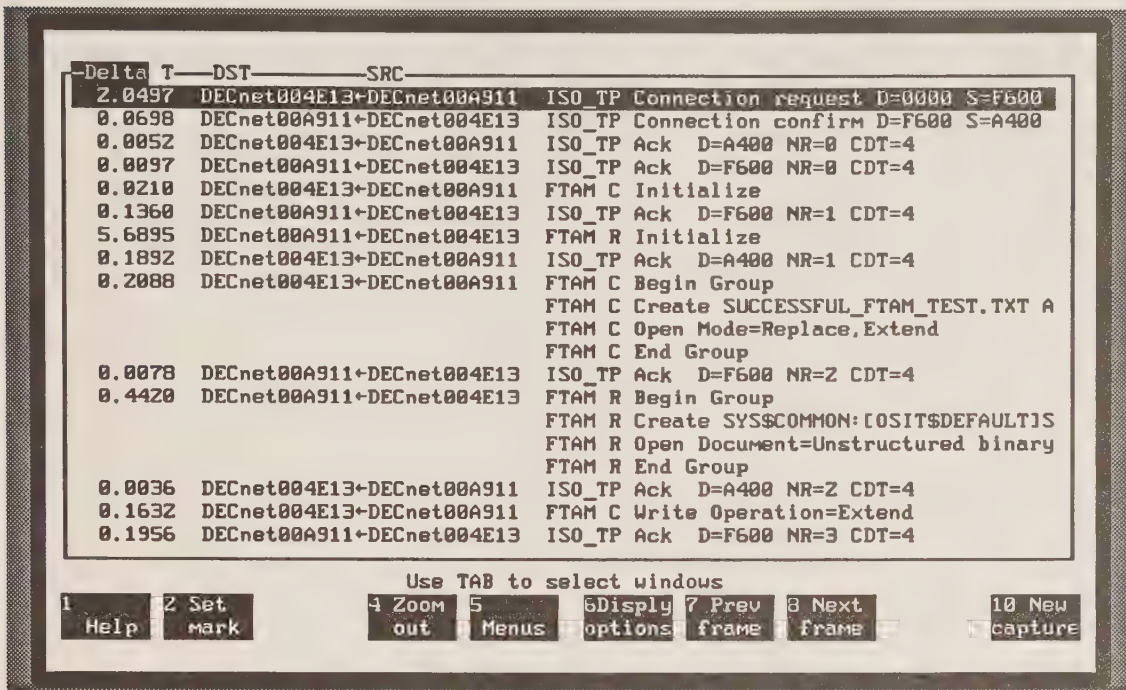
- FTAM-1: unstructured text files
- FTAM-2: sequential text files
- FTAM-3: unstructured binary files
- FTAM-4: sequential binary
- FTAM-5: simple hierarchical

The FTAM model is based on a single file. Most operating systems have more than one file, structured as a directory. A standard developed by the National Institute of Standards and Technology (NIST, formerly known as the National Bureau of Standards) provides a supplemental document type, that treats files in a directory as a special type of file. The user can retrieve this special file then move on to individual files.

Figure 14-11 shows an example of a typical FTAM session. The first thing that happens is that an OSI transport protocol virtual circuit is established and the connection request and confirm packets are acknowledged.

Once the circuit is initialized, the FTAM initialize message is sent, presumably embedded within OSI ACSE, Presentation, and Session Layer initialize messages. Of course, it is possible to have the session layer reuse an existing transport layer connection. If so, there is no need to initialize. This initialize message is also acknowledged at the underlying transport layer, as is the initialize response.





14-11 FTAM Traffic

Once the FTAM association is in place, the requesting node sends a series of operations, structured together as a single group of operations. First, a file is created, and then the file is opened in “replace, extend” mode. Replace, extend mode is used in FTAM document types 1 and 3; for document type 2, the insert mode is used instead.

Next, the other node sends a message back acknowledging each of the operations. Notice that the file name returned is the full name which has been translated into an open operation on the unstructured binary document type. The first node then goes back and extends its file.

Figure 14-12 shows the FTAM initialize request in more detail. The service is being established based on functional units. The node has requested write, limited file management, and grouping. Attribute groups signify which file attributes are negotiated for a session.

The FTAM quality of service is used to indicate if recovery is needed on this operation. The recovery and underlying checkpointing can make the session significantly slower, so if the application doesn’t need it, it is often avoided.

The initialize request also indicates that two document types maybe used, the unstructured and sequential text document types. It is up to each side to know what the document types mean in their real file systems and how they are then interpreted by the application.

Figure 14-13 shows a little more detail on the operation to close and deselect a file. In between each of the FTAM commands, there is a presentation layer header which indicates which of the possible presentation context identifiers apply to the upcoming command.

Figure 14-14 shows a session with a data transfer operation. First a file is created, and the file extended. Next, 160 bytes of data are transferred, followed by a data end (the end of the data regime) and then a transfer end (the end of the transfer regime). Finally, the file is closed and deselected, freeing it up for other users.

Figure 14-15 shows another simple session, in which a file is created and opened as an unstructured text document. There are ten groups of text data sent, all included in the same underlying Ethernet packet. The text data is followed by a data end message and an end of the transfer regime.

One of the aspects of FTAM that makes it interesting is the large number of file attributes that have been defined (see Fig. 14-16). Attributes are broken up into four groups, and not all groups must be supported.

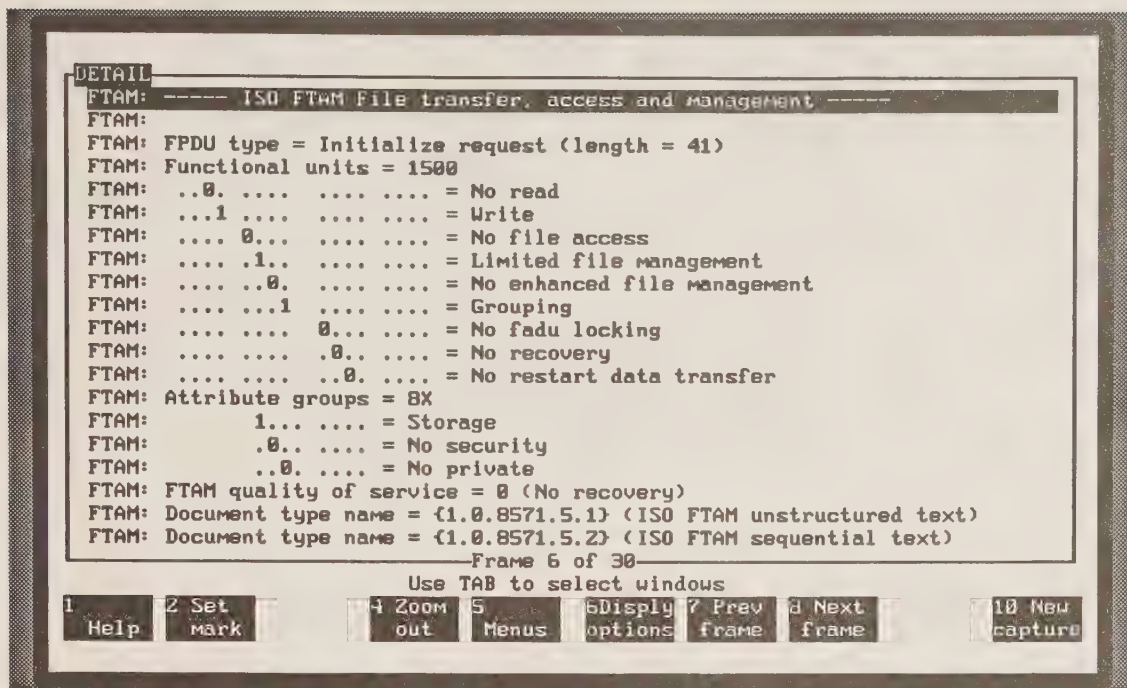
The kernel has the basic information on a file, including the name and permitted action. The contents type indicates document type (or a more general classification called a constraint set) of the file. Note that just because this information exists doesn't mean that the operating system will necessarily allow it to be divulged to other users.

The storage group includes an account indicator so that charges for access to the file may be charged to an account. The storage group also indicates size and availability information (e.g., online or on tape) as well as date and time information. The security group is used for a variety of purposes. The access control indicates who may look at the data in the file (or perform other operations such as writing). The legal qualifications indicates information such as copyright that may reflect how the data may be used.

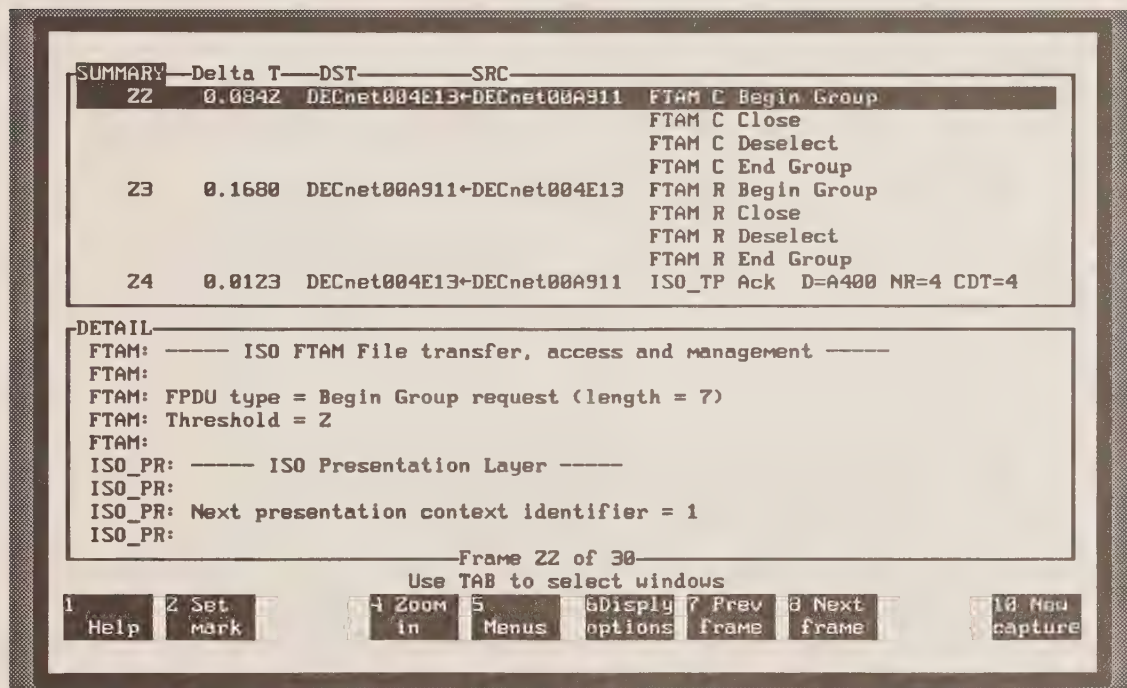
Finally, there is a group called the private group, which has a single private attribute. This is where a particular implementation or computing environment would put historical file attributes or information that is not supported in FTAM.

### *Sun's FTAM Implementation*

Sun supports FTAM, as most vendors do, as a gateway into their native environment. SunLink OSI includes a basic FTAM implementation. The gateway sits at the edge of the network and is able to establish FTAM sessions to OSI networks. A user simply accesses the OSI-based service via the gateway system. Chapter 16 on interoperability discusses this issue in a bit more detail.



14-12 An Initialize Request



14-13 Close and Deselect Operations



SUMMARY	Delta T	DST	SRC	
6	2.1557	DECnet004E13+DECnet00A911	FTAM C Initialize	
8	5.8255	DECnet00A911+DECnet004E13	FTAM R Initialize	
10	0.3981	DECnet004E13+DECnet00A911	FTAM C Begin Group	
			FTAM C Create SUCCESSFUL_FTAM_TE	
			FTAM C Open Mode=Replace,Extend	
			FTAM C End Group	
12	0.4499	DECnet00A911+DECnet004E13	FTAM R Begin Group	
			FTAM R Create SYS\$COMMON:[OSIT\$D	
			FTAM R Open Document=Unstructure	
			FTAM R End Group	
14	0.1668	DECnet004E13+DECnet00A911	FTAM C Write Operation=Extend	
16	0.7753	DECnet004E13+DECnet00A911	FTAM Binary Data [160 bytes]	
18	0.0261	DECnet004E13+DECnet00A911	FTAM C Data End	
19	0.0242	DECnet004E13+DECnet00A911	FTAM C Transfer End	
21	0.0963	DECnet00A911+DECnet004E13	FTAM R Transfer End	
22	0.0842	DECnet004E13+DECnet00A911	FTAM C Begin Group	
			FTAM C Close	
			FTAM C Deselect	
			FTAM C End Group	
23	0.1680	DECnet00A911+DECnet004E13	FTAM R Begin Group	

Use TAB to select windows

1 Help	2 Set mark	4 Zoom out	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	------------	---------	-------------------	--------------	--------------	----------------

14-14 Data Transfer Request

SUMMARY	Delta T	DST	SRC	
13	0.5022	DECnet00A911+DECnet004E13	FTAM R Begin Group	
			FTAM R Create SYS\$COMMON:[OSIT\$D	
			FTAM R Open Document=Unstructure	
			FTAM R End Group	
15	0.1829	DECnet004E13+DECnet00A911	FTAM C Write Operation=Extend	
17	1.3213	DECnet004E13+DECnet00A911	FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [140 bytes]	
			FTAM Text Data [108 bytes]	
20	0.0217	DECnet004E13+DECnet00A911	FTAM C Data End	
22	0.0272	DECnet004E13+DECnet00A911	FTAM C Transfer End	
23	0.1156	DECnet00A911+DECnet004E13	FTAM R Transfer End	
24	0.0814	DECnet004E13+DECnet00A911	FTAM C Begin Group	
			FTAM C Close	

Use TAB to select windows

1 Help	2 Set mark	4 Zoom out	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	------------	---------	-------------------	--------------	--------------	----------------

14-15 File Attributes



Kernel Group	File name
	Permitted action
	Contents type
Storage Group	Storage account
	Date/Time attributes
	IDs
	File availability
	File size
Security Group	Future file size
	Access control
	Legal qualifications
Private Group	Current access passwords
	Private attributes

**14-16**  
FTAM Attributes

## Resource Location

Finding resources on a network is becoming an increasingly difficult task. This question is discussed in several places in this book. In Chapter 15, “Network Management,” we see the Discover Tool, a part of SunNet Manager. In Chapter 9, “Names,” we looked at three directory services (NIS+, NIS, and DNS) and touched on two more (X.500 and DEC DNS). Here, we look at two more services, these specifically tailored for the problem of finding people. There is nothing more frustrating than being unable to communicate with a person because a properly formed address—phone number, electronic mail address, or fax number—can’t be found.

Eventually, services like X.500 will provide a solution to the problem of finding people. However, for several years the whois and finger services have provided a rudimentary source of knowledge.

### *The whois Protocol*

The Domain Name System is fine for resolving machine-friendly things like domain names to server IP addresses, but this is a bit low level for many humans. The whois service is provided as a TCP service from the Network Information Center (the host SRI-NIC provides this service and runs on IP addresses 26.0.0.73 or 10.0.0.51). A simple request/response protocol, the service accepts a search parameter and returns four pieces of information:

- Full name
- United States mailing address
- Telephone number

SMITH	Look for name or handle Smith.
!SRI-NIC	Look for handle SRI-NIC only.
.Smith	Look for last name Smith.
Mal...	Look for anything starting with Mal.
smith@	Look for Smith at unknown host.
@host	All users on known host.
smith@host	Look for username Smith on host.
*SRI-NIC	Find all names ending in SRI-NIC.

## 14-17 The whois Service

- Network mailbox users

It is up to individuals to register themselves in the database. The Defense Communications Agency, who runs the DoD portion of the Internet, requests all individuals who pass traffic across the network to register themselves. Registration is simple. A user sends electronic mail to:

registrar@nic.ddn.mil

The mail should include full name, middle initial, United States mailing address (including mail stop and full explanation of abbreviations and acronyms), ZIP code, telephone, and one network mailbox.

When the service is accessed, the user submits a single name specification (see Fig. 14-17). If the name specification is a question mark, the service sends back a list all possible formats.

### *The Finger Protocol*

Many Unix systems have a finger command which is used to find out information about users and hosts throughout the network. The program that is queried is known as a remote user information program (RUIP). RFC 1196 defines a protocol to be used between a user (e.g., the finger command) and a RUIP.

This protocol uses TCP port 79. The user sets up a connection to this port, which signals the RUIP to become available to process requests. The protocol consists of a one-line query. The RUIP responds with information and closes the connection. All data transferred in this query is simple ASCII data.

Finger provides for two types of queries. In the first type, the query is the name of a remote user. The answer is information about the user. The second form is a bit exotic and consists of a list of hosts to forward the query to, a form of source routing. Normally, a query is something like:

myhost% finger user2@host2

This results in a connection being set up to host2 and the finger command “user2” is sent. The answer is some host-dependent information about that user. A wordy (/w) flag asks for more verbose information to be sent.

A special form of this query consists of a null username. This is asking for information about all online users. The remote RUIP has the choice of answering or not.

The exotic form is a series of hosts. The command is sent to the first host in the list, which is asked to forward the query to another RUIP. The first RUIP must either actively accept or reject this command; it is not allowed to simply ignore it.

If accepted, the query is forwarded until the final host is received. Then, the results are passed back and the answer presented to the user.

When a query is accepted, the answer consists of at least the user’s human name. Additional information which might be included could be the terminal location, the office location, the office phone number, the job title, or the amount of idle time on the user’s account (indicating if the user is actively working on the terminal or just using up a slot).

Most systems include an additional .plan file that each user can keep. Inside that file is information that the user wants presented to the outside world. It can have the current projects that the user is working on but can also have special information (e.g., “If I’m not in my office, don’t call me at home.”).

It is possible that the username received is ambiguous, as in the case of sending the username William and there are WilliamA and WilliamB usernames on the systems. It is up to the RUIP to figure out how to handle this. It can reject the request as ambiguous, look for all longer names, or even run a soundex on the input to see if any usernames “sound like” the string supplied (a useful feature if your name is often misspelled).

The finger protocol is, by definition, a security risk. You are telling people in the outside world about your environment. It is up to the local network administrator or an administrator on a host to decide if finger should be run at all, and if so, what information to put into responses.

Some information that is often presented is when the last mail message was read by a user and if there is any unread mail waiting. Some systems even go so far as to indicate who the most recent mail was from. This information can be used to track the flow conversations.

There is a possible security problem on the client side too. A malicious finger program (or malicious user with a .plan file) can include things like escape sequences that will create havoc with the user’s screen. A good finger client allows the user to filter out unprintable characters.

A final note on the finger protocol helps bring out an important point: users are not necessarily people. There are several implementations of the finger protocol which give information on nonhuman entities. For exam-



ple, there are at least two computing sites, `weather@madlab.sprl.umich.edu` and `weather@groucho.ucar.edu` that have weather built servers on the finger protocol (see Fig 14-18).

There are even rumors of Coke machines that inform the network of their current contents using the protocol. According to David Zimmerman of Rutgers, author of the finger RFC, "There is the historical story of the CMU Coke machine—upon fingering `coke@???cmu.edu` (I forget what the name of ??? was), you would get information about its contents."

## Trivial FTP

Trivial FTP (TFTP) is a simpler version of FTP. It is used to read or write files to and from a remote server. TFTP might be used to move mail messages back and forth, and it forms the way that a diskless node gets its operating system over the network.

TFTP has three modes of data transfer:

- netASCII
- Octet
- netASCII to a user instead of a file (e.g., mail)

All transfers begin with request to read or write file. This request also sets up the connection. If server grants the request, data transfer is in blocks of 512 bytes. Any block less than this (0 to 511 bytes) is the end of transfer.

The connection requestor sends a read request (RRQ) or write request (WRQ) and receives back a positive acknowledgment. Following that, each data packet gets a block number. Block numbers start at 1 and are consecutive. Block number 0 is the acknowledgment packet to an initial write request.

All data blocks are explicitly ACKed. Lock step acknowledgment makes this a simple protocol. Most errors cause termination via an error packet. The error packet is not ACKed.

TFTP is based on UDP, using well-known port number 69. The connection is identified by the UDP-level transaction IDs (TIDs). The initial request has a source TID and is sent to well-known port 69 decimal (105 octal). The response to the initial request will have a different TID chosen by the server process. The resulting source/destination TID will be used for subsequent traffic. TIDs should be chosen at random so there is little chance of the same number being used in near succession.

Simple? That's the whole idea. There are only four packet types. Read and write requests get things going. Data packets and acknowledgments transfer the data. Receiving anything less than a full block terminates the session. This simplicity is why TFTP is used for remote booting—it is simple enough to place the protocol into a ROM chip.



```

rutgers$ finger weather@madlab.sprl.umich.edu
(madlab.sprl.umich.edu)
Login name: weather           In real life: Home of the Weather Re-
port
Directory: /sdm/weather      Shell: /bin/false
Never logged in.
New mail received Thu Nov 15 15:53:05 1990;
  unread since Fri Sep 14 13:39:56 1990
Plan:

```

```

614
FQUS1 KDTW 130325
MIZ028030-131010-

```

```

DETROIT METROPOLITAN AREA FORECAST
NATIONAL WEATHER SERVICE DETROIT MI
1030 PM EST SAT JAN 12 1991

```

```

.TONIGHT...VARIABLE CLOUDINESS. COLDER...LOW AROUND 20. LIGHT
AND VARIABLE WIND.
.SUNDAY...MOSTLY SUNNY. HIGH IN THE MIDDLE 30S. WIND BECOMING
SOUTHWEST 5 TO 15 MPH.
.SUNDAY NIGHT...INCREASING CLOUDINESS...LOW IN THE MIDDLE 20S.
.MONDAY...CLOUDY WITH A 30 PERCENT CHANCE OF SNOW. HIGH IN
THE MIDDLE 30S.

```

GHW

```

rutgers$ finger weather@groucho.ucar.edu
(unidata.ucar.edu)
Login name: weather           In real life: Denver Metro Forecast
Directory: /data/weather      Shell: /data/weather/dbq
Last login Fri Jan 11 13:28 on ttyb from CSLI.Stanford.ED
No unread mail
Plan:

```

```

914
FQUS1 KDEN 130435
COZ011-131100-

```

```

DENVER METROPOLITAN FORECAST

```

Read Request (RRQ) and Write Request (WRQ) Packets	
opcode	rrq=1 (2 bytes) wrq=2
file name	
terminator	1 byte = 0
mode	Mode is netascii, octet, or mail. If mail, then the file name is a username. Mail mode is always a WRQ.
terminator	1 byte = 0
Data Packet	
opcode	data = 3
block #	2 bytes
Data	512 bytes: normal packet EOT: 0-511 bytes
ACK Packet	
opcode	ACK = 4
block #	(2 bytes)
Error Packet	
opcode	opcode = 5
error code	2 bytes
error string	
terminator	1 byte = 0

#### 14-19 Trivial FTP Packets

Figure 14-19 shows the entire TFTP protocol, consisting of four different kinds of packets. Figures 14-20 and 14-21 shows some TFTP traffic on a network. In Figure 14-20, the exchange begins with a simple read file request for the file config.sys. Obviously, this is a PC booting itself up. Figure 14-21 shows a simple write request. Notice that just before the TFTP write request goes through, there is an ARP request to find out the IP address of the TFTP server.

#### Miscellaneous TCP/IP Services

Several useful utilities were defined by Jon Postel in some 1983 RFCs. The echo server is a TCP-based service that listens on TCP port 7. When a connection is set up, any data sent to the server is sent back until the calling user terminates the connection. The UDP version of the service (also on port 7) simply sends back a datagram.

SUMMARY	Delta T	DST	SRC	
3		[128.1.0.1]	+ [128.1.0.2]	TFTP Read request File=\config.s
4	2.0766	[128.1.0.2]	+ [128.1.0.1]	TFTP Data packet NS=1 (Last)
5	0.0389	[128.1.0.1]	+ [128.1.0.2]	TFTP Ack NR=1
8	614.3770	[128.1.0.1]	+ [128.1.0.2]	TFTP Read request File=\config.s
9	1.5479	[128.1.0.2]	+ [128.1.0.1]	TFTP Data packet NS=1 (Last)
10	0.0391	[128.1.0.1]	+ [128.1.0.2]	TFTP Ack NR=1
13	293.5331	[128.1.0.1]	+ [128.1.0.2]	TFTP Write request File=\junk.tx
14	4.3785	[128.1.0.2]	+ [128.1.0.1]	TFTP Ack NR=0
15	0.0673	[128.1.0.1]	+ [128.1.0.2]	TFTP Data packet NS=1 (Last)

## DETAIL

```

TFTP: ----- Trivial file transfer -----
TFTP:
TFTP: Opcode = 3 (Data packet)
TFTP: Block number = 1
TFTP: [76 bytes of data] (Last frame)
TFTP:
TFTP: [Normal end of "Trivial file transfer".]
TFTP:

```

Frame 4 of 22

Use TAB to select window

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

14-20 Last Packet of a TFTP Transfer

SUMMARY	Delta T	DST	SRC	
10	0.0391	WstDigE58C11+WstDig488C11		TFTP Ack NR=1
11	293.4669	Broadcast	+WstDig488C11	ARP C PA=[128.1.0.1] PRO=IP
12	0.0357	WstDig488C11+WstDigE58C11		ARP R PA=[128.1.0.1] HA=0000C0E5
13	0.0304	WstDigE58C11+WstDig488C11		TFTP Write request File=\junk.tx
14	4.3785	WstDig488C11+WstDigE58C11		TFTP Ack NR=0
15	0.0673	WstDigE58C11+WstDig488C11		TFTP Data packet NS=1 (Last)
16	0.0486	WstDig488C11+WstDigE58C11		TFTP Ack NR=1
17	127.8132	Broadcast	+WstDig488C11	ARP C PA=[128.1.0.1] PRO=IP
18	0.0295	WstDig488C11+WstDigE58C11		ARP R PA=[128.1.0.1] HA=0000C0E5

## DETAIL

```

TFTP: ----- Trivial file transfer -----
TFTP:
TFTP: Opcode = 2 (Write request)
TFTP: File name = "\junk.txt"
TFTP: Mode = "netascii"
TFTP:
TFTP: *** 10 byte(s) of additional data present ***
TFTP:
TFTP: [Abnormal end of "Trivial file transfer".]

```

Frame 13 of 22

Use TAB to select window

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

14-21 A TFTP Write Request



Code	Description
1	Print any awaiting jobs
2	Receive a job
3	Send queue state (short listing)
4	Send queue state (verbose)
5	Remove jobs
6	Receive job substate
Receive Job Subcommands	
1	Abort
2	Receive control file
3	Receive data file

14-22

LPR Commands

The discard protocol takes any data it receives and throws it away. It continues as long as the connection is active if it based on TCP (port 9) or for a single datagram for the UDP port 9.

The character generator, on port 19, sends data without regard to input. Data can be anything, but it is recommended that it have a recognizable pattern. Typically, the protocol takes the 95 printing ASCII characters and prints them 72 characters to a line to form a repeating pattern.

The daytime protocol (port 13) prints the date and time in ASCII format. While daytime prints a user-friendly time, the time protocol (port 37) is meant to send a machine-friendly version. Time is defined as the number of seconds since midnight of January 1, 1900, represented as a 32-bit number.

The daytime and time protocols are primitive predecessors of the Network Time Protocol. The older protocols provide a machine's version of the time, whereas NTP synchronizes the clocks of many computers with some primary reference source.

## The Line Printer Daemon

Line printer (lpr) is a Berkeley-developed set of commands to submit jobs to a remote computer and to manage the print queue. The protocol consists of four user commands on the client machine and a line printer daemon on the server. The four user commands are:

- lpr: assign a job to a queue
- lpq: display jobs in a queue
- lprm: remove a job from a queue
- lpc: control a queue



Code	Name	Description
C	Class for banner page	This command sets the class name to be printed on the banner page. The name must be 31 or fewer octets. The name can be omitted. If it is, the name of the host on which the file is printed will be used. The class is conventionally used to display the host from which the printing job originated. It will be ignored unless the print banner command ("L") is also used.
H	Host name	This command specifies the name of the host which is to be treated as the source of the print job. The command must be included in the control file. The name of the host must be 31 or fewer octets.
I	Indent printing	This command specifies that, for files which are printed with the "f" command, indent to the columns given. (It is ignored for other output generating commands.) The indenting count operand must be all decimal digits.
J	Job name for banner page	This command sets the job name to be printed on the banner page. The name of the job must be 99 or fewer octets. It can be omitted. The job name is conventionally used to display the name of the file or files which were "printed." It will be ignored unless the print banner command ("L") is also used.
L	Print banner page	This command causes the banner page to be printed. The username can be omitted. The class name for banner page and job name for banner page commands must precede this command in the control file to be effective.
M	Mail when printed	This entry causes mail to be sent to the user given as the operand at the host specified by the "H" entry when the printing operation ends (successfully or unsuccessfully).
N	Name of source file	This command specifies the name of the file from which the data file was constructed. It is returned on a query and used in printing with the "p" command when no title has been given. It must be 131 or fewer octets.
P	User identification	This command specifies the user identification of the entity requesting the printing job. This command must be included in the control file. The user identification must be 31 or fewer octets.
S	Symbolic link data	This command is used to record symbolic link data on a Unix system so that changing a file's directory entry after a file is printed will not print the new file. It is ignored if the data file is not symbolically linked.

Code	Name	Description
T	Title for pr	This command provides a title for a file which is to be printed with either "p" command. (It is ignored by all of the other printing commands.) The title must be 79 or fewer octets.
U	Unlink data file	This command indicates that the specified file is no longer needed. This should only be used for data files.
W	Width of output	This command limits the output to the specified number of columns for the "f,," "l,," and "p" commands. (It is ignored for other output generating commands.) The width count operand must be all decimal digits. It may be silently reduced to some lower value. The default value for the width is 132.
1	Troff R font	This command specifies the file name for the troff R font. This is the font which is printed using Times Roman by default.
2	Troff I font	This command specifies the file name for the troff I font. This is the font which is printed using Times Italic by default.
3	Troff B font	This command specifies the file name for the troff B font. This is the font which is printed using Times Bold by default.
4	Troff S font	This command specifies the file name for the troff S font. This is the font which is printed using Special Mathematical Font by default.
c	Plot CIF file	This command causes the data file to be plotted, treating the data as CIF (Caltech Intermediate Form) graphics language.
d	Print DVI file	This command causes the data file to be printed, treating the data as DVI (TeX output).
f	Print formatted file	This command causes the data file to be printed as a plain text file, providing page breaks as necessary. Any ASCII control characters which are not in the following list are discarded: HT, CR, FF, LF, and BS.
g	Plot file	This command causes the data file to be plotted, treating the data as output from the Berkeley Unix plot library.
k		Reserved for use by Kerberized LPR clients and servers.
l	Print file leaving control characters	This command causes the specified data file to printed without filtering the control characters (as is done with the "f" command).

Code	Name	Description
n	Print ditroff output file	This command prints the data file to be printed, treating the data as ditroff output.
o	Print PostScript output file	This command prints the data file to be printed, treating the data as standard PostScript input.
p	Print file with "pr" format	This command causes the data file to be printed with a heading, page numbers, and pagination. The heading should include the date and time that printing was started, the title, and a page number identifier followed by the page number. The title is the name of file as specified by the "N" command, unless the "T" (title) command has been given. After a page of text has been printed, a new page is started with a new page number. (There is no way to specify the length of the page.)
r	File to print with FORTRAN carriage control	This command causes the data file to be printed, interpreting the first column of each line as FORTRAN carriage control. The FORTRAN standard limits this to blank, "1,," "0,," and "+" carriage controls. Most FORTRAN programmers also expect "-" (triple space) to work as well.
t	Print troff output file	This command prints the data file as Graphic Systems C/A/T phototypesetter input. This is the standard output of the Unix troff command.
v	Print raster file	This command prints a Sun raster format file.
z		Reserved for future use with the Palladium print system.
Source: RFC 1179		

## 14-23 (Cont.) LPR Control File

Note that this protocol does not control how a job gets printed: that is the job of some local program. All the lpr commands do is get a job into a place where the local program will be likely to find it.

The lpr daemon is a TCP-based service that listens on port 515. The protocol is quite simple. Each command begins with a 1-byte command code, followed by the ASCII name of the printer queue, followed by some operand.

Each command to the daemon requires a new connection. Most commands are a very simple request/response protocol, followed by the connection being dismantled. Figure 14-22 shows the commands defined for LPR.

The procedure consists of first transferring over a control file, which has a set of print options and commands in it. Next, the data file (the file to be

printed) is transferred over. All control files must be named starting with the letters DFA, a 3-digit job number, and the name of the submitting host.

After the control file is successfully sent, the data file is sent. This file starts with the letters DFA, the same 3-digit job number as the control file, and (one hopes) the same host name.

The control file consists of a series of commands. Each line has a 1-character command and any operands. Figure 14-23 shows the possible commands: note that they are case sensitive. This file must have at least three lines in it. The H command identifies the responsible host, and the P command is the responsible user. There must then be at least one lowercase command to produce output.

It doesn't get any simpler than this.

### For Further Reading

K. Harrenstein, M. Stahl, E. Feinler, *NICNAME/WHOIS*, RFC 954, October, 1985.

E. Krol, *The Hitchhikers Guide to the Internet*, RFC 1118, September 1989.

L. McLaughlin III, *Line Printer Daemon Protocol*, RFC 1179, August 1990

J. Postel, *Character Generator Protocol*, RFC 864, May, 1983.

———, *Daytime Protocol*, RFC 867, May, 1983.

———, *Discard Protocol*, RFC 863, May, 1983.

———, *Echo Protocol*, RFC 862, May, 1983.

J. Postel, K. Harrenstien, *Time Protocol*, RFC 868, May, 1983.

J. Postel, J. Reynolds, *File Transfer Protocol (FTP)*, RFC 959, October 1985

———, *Telnet Protocol Specification*, RFC 854, May 1983.

Barry Shein, "The Telnet Protocol," *ConneXions*, v. 3, no. 10, October 1989, pp. 32-38.

K. R. Sollins, *The TFTP Protocol (Revision 2)*, RFC 783, June, 1981

James VanBokkelen, "Workstation Oriented Enhancements to Telnet," *Connexions*, v. 3, no. 7, July 1989.



## CHAPTER 15

# Management



# Management

## Managing a Network

There are three elements to a network management system:

- Manager
- Agent
- Managed object

The managed object is some module on the network. It could be a Unix kernel, a routing module (such as `routed`), a name server, a vending machine, or any of the other objects of interest on a network. In fact, the definition of an object transcends the boundaries of a host, as in the case of an Ethernet cable or a collection of dumb devices. A human wants to be able to observe and control any of these objects.

The manager is a piece of software (sometimes known as a director) that issues management requests on behalf of the human. Note that many of these requests may be programmed, as in the case of the human informing the director that he or she is interested in a certain piece of data every 10 minutes.

The agent is the piece of software on a remote node that is able to respond to manager requests and translate them into some form of internal call to the object being managed. The language used between the agent and the manager is a network management protocol. The Simple Network Management Protocol (SNMP) and the RPC-based management services are two languages that can be used.

The managed object may or may not be aware of the manager. One of the advantages of the agent/manager model is that the agent can field requests from many different managers on many different nodes. The agent is then responsible for figuring out how to control the actual module or object being managed.

The agent might issue a directive to the managed object, but often the interaction is more subtle. Objects have a set of attributes, some of which contain data for the status of the object. Number of bytes transmitted is a status attribute for an Ethernet card.

Other attributes, however, govern the characteristics of the object. Rather than control the object directly, we merely change one of these characteristic attributes, so that the object will notice the change and thus alter its behavior.

This chapter begins with the definition of what is being managed. A standard definition of a management information base (MIB) means that different vendors products, if they have the same basic functionality, can all be controlled by a single manager (or by multiple managers from different vendors).

There are really two pieces to defining the things being managed. First, there needs to be a way to name the things being managed. This is called the structure of management information (SMI). Second, there are definitions of the objects themselves, defined in a variety of MIB RFCs. The core objects for the TCP/IP stack are defined in the MIB II (the second revision of the core MIB). Other MIBs define specialized devices, such as T1 ports, serial ports, and even toasters.

Once the definition of objects is discussed, this chapter examines how to communicate requests to those objects. This is the role of the Simple Network Management Protocol (SNMP), a standard TCP/IP-based language for communication between managers and agents (who then communicate with the objects).

Third, we look at the platform used in Sun networks to set up different kinds of managers, the SunNet Manager. SunNet Manager manages SNMP objects but can also use SunNet Manager calls built on top of RPC as a language for managing objects.

It should be noted that there are other approaches to this problem. DEC, for example, has specified the CMIP language instead of the SNMP language as the native interface for DECnet Phase V. They have a different architecture for the manager, known as DECMcc. However, CMIP implementations come in several proprietary flavors, whereas SNMP-based approaches are standard and widely available.

Managing across architectures is not an easy problem. While SunNet Manager could be used to manage both TCP/IP and DECnet environments (using a NICE agent to speak the native DECnet management protocol), there are always some advantages to using the native protocol. One strategy is to try and make a single protocol the choice on a network. If that is not possible, many environments end up with several different management tools.



There are some attempts being made to provide for manager-to-manager communication protocols. Manager, in this case, refers to software modules and not people in suits. There is something to be said for improving communication between people, but that's the subject of a different book.

## Structure of Management Information

The “structure and identification of management information” is a fancy term for a standard way to name objects that will be managed. RFC 1155, which has this imposing title, defines a registration procedure for managed objects. Since the number of objects and their attributes can be very large, a standard method is essential.

The naming includes two basic pieces. First, there is a method for identifying objects, using a hierarchical name tree similar to other naming systems described in this book. Second, there are some standard data types used to describe what these objects are.

The structure document is thus a metaschema. It does not define any actual objects, only how to define objects. The objects themselves are defined in the MIBs. Accessing the objects requires a knowledge of SNMP plus some knowledge of the underlying MIB (although we will see that it is possible to “walk the tree” without knowing everything about all objects).

Every object that is managed has a name, syntax, and encoding. The name is uniquely represented by an object identifier. Object identifiers are administratively assigned names. The syntax is a highly restricted subset of ASN.1. The encoding is simply how the instances of that object type are represented. The syntax and encoding are closely tied to transmission over the network: SNMP uses a subset of the Basic Encoding Rules (BER) that accompany ASN.1.

## Names

Names are hierarchical in nature. An identification model is used which assigns an integer number to objects at each level. An object identifier is thus a sequence of integers which traverse a global tree. To help humans, each of these object identifiers is paired with a name. It is possible for each object to have a subtree. If we have an object named database for example, it might have subobjects named tables and users. The subobjects, in turn, can have their own subobjects.

As one traverses the tree to lower and lower levels, it is possible to delegate administrative control, just as we saw with the Domain Name System. The top of the tree is administered by the Internet Assigned Numbers Authority. Lower levels might be administered by a particular vendor or user.

The root of this tree is unlabeled but has at least three children: ISO (1), CCITT (0), and joint ISO/CCITT (2). An object identifier starting with a 1 is thus under the control of ISO. Any number under 2 is under the joint control of dueling bureaucracies.

Underneath the ISO tree there is an object identifier 3 for organizations, under which is the U.S. Department of Defense, which has a label of 6. To refer to the object DoD, the object identifier 1.3.6 is used.

DoD ceded a portion of its tree to the Internet. The object Internet is thus defined as:

```
internet OBJECT IDENTIFIER ::=
    { iso org(3) dod(6) 1 }
```

The Internet Activities Board has divided the space up further into four pieces:

directory	internet 1
mgmt	internet 2
experimental	internet 3
private	internet 4

The directory subtree is reserved for a future memo that discusses how to use the OSI directory in the Internet. Mgmt is used for documents in IAB-approved standards. Administration of this subtree is delegated to the Internet Assigned Numbers Authority for the Internet.

For example, the original MIB was issued as an RFC and was assigned the number:

```
1.3.6.1.2.1
```

Experimental is also managed by the Assigned Numbers Authority. Finally there is private (also delegated by assigned numbers). The private subtree has a subnode that is delegated for enterprises:

```
enterprises OBJECT IDENTIFIER ::= { private 1 }
```

Enterprises allows vendors to register models of their products (the registrar is Joyce K. Reynolds at ISI, reachable over electronic mail at [jk-rey@isi.edu](mailto:jk-rey@isi.edu)). It is recommended that each vendor register its products here so there is no future conflict.

```
companya ::= { enterprises 2 }
product1 ::= { companya 3 }
```

At this point, we have a mechanism for assigning object identifiers. In addition to naming objects, we also want to be able to refer to attributes of that object (which are also given identifying numbers lower in the tree).

NetworkAddress	Choice (currently only the Internet family).
IpAddress	Octet string of length 4.
Counter	32-bit unsigned, monotonically increasing integer (may wrap).
Gauge	32-bit unsigned integer which increases or decreases but latches at a maximum value.
TimeTicks	Counts time in hundredths of a second.
Opaque	An arbitrary value encoded as a string of octets and then double-wrapped.

### 15-1 Defined Data Types in SNMP

A basic set of primitive data types have been defined to be used for most MIB definitions. Keeping the data types fairly simple means that most managers should be able to look at most types of objects. There are several primitive data types: integer, octet string, object identifier, and null. There are also several defined data types such as the six types defined in the SMI document (see Fig. 15-1). An informal convention for enumerated integers is that the value of 0 is reserved and should not be used. There are also two constructed data types. A list is represented as a sequence of primitive types. A table is a sequence of lists.

When the data is actually sent over the network, a subset of the Basic Encoding Rules (BER) of ASN.1 are used (as opposed to XDR). The ASN.1 Basic Encoding Rules have the advantage that they are self-describing: a node can tell what kind of data it is getting from the data itself.

In order to define an object, there are five pieces of information needed:

- Textual name (the object descriptor) plus object identifier
- Syntax
- Definition
- Access
- Status

A sixth element, an index, is used for multi-instance variables. Access specifies if this object is available to managers and agents. Standard access classes are read-only, read-write, write-only, or not accessible. Note that the definition of access inside a MIB is different from access control provided by SunNet Manager objects.

The status of an object indicates its status as an Internet standard. The four status indicators are mandatory, optional, deprecated (i.e., not recommended, but still hanging on), and obsolete. Mandatory means that if there is a product that fits into the general class (e.g., a TCP implementation), the object must be provided.



Obsolete is actually a fairly useful status to have. Rather than deleting an object, it is simply marked obsolete. A rule is that there can't be a new object that duplicates the name of an obsolete object. This means that a manager knows when it finds a particular object identifier what that object means.

There is a difference between object types (defined in a MIB) and object instances. An object instance is an instantiation of an object type which has been bound to a value. The means by which object instances are referenced is part of the concise MIB format index clause. A protocol like SNMP provides a means for accessing simple object types. It also provides simple tools that are used to perform aggregate operations like bulk retrieval or walking the tree. In SNMP, this is handled by the get-next operator.

Object identifiers can be identified with a version number. New versions can declare old object types obsolete but not delete their names. New versions can also augment the definition of an object type in a list by appending other nonaggregate object types to the list or by defining new object types, but may not change the semantics of any previously defined object without changing the name of that object.

## SNMP

The Simple Network Management Protocol (SNMP) is, as its name promises, a simple way for two nodes to communicate management information. An in-depth discussion of the protocol can be found in the defining RFCs or in Marshall Rose's *The Simple Book*.

For our purposes, we will show the SNMP operation with four sample screens from a network (see Figs. 15-2 through 15-5). The basic operation is a get request, shown in Figure 15-2. The request is identified with a request ID so that answers coming back can be correlated.

Notice that the packet has both error flags and error index values set at zero, since this is an outgoing request. The community is Interop, indicating that the packet was captured during the annual INTEROP trade show in San Jose. Notice that subsequent packets in Figure 15-2 show that the data was returned and that a get-next message was then issued. More on the get-next command in a minute.

Figure 15-3 shows the answer to the previous request. Notice that the request ID matches that of the previous figure. The error is a "no such name" error, indicating that the device that was queried had no variable with that identification number (remember that names are just a shorthand for the SMI-based object ID).

Figure 15-4 shows the get-next command. An object ID can be furnished and a request can be set for the next object ID based on walking the object ID hierarchy. This command can specify several different objects, typically



SUMMARY	Delta T	DST	SRC	
19	0.0053	DEC	07F2F1+Sun	01E954 SNMP Get ipRouteNextHop
20	0.1669	Intrln042DF4+Gould	000026	Telnet C PORT=2355 rm d
21	0.1128	Sun	00394E+DEC	07F2F1 SNMP Got tcpConnRemAddress ... t
22	0.0068	DEC	07F2F1+Sun	00394E SNMP Next tcpConnRemAddress ...
23	0.0465	Sun	01E954+DEC	07F2F1 SNMP Got No such name ipRouteNex
24	0.0048	DEC	07F2F1+Sun	01E954 SNMP Get ipRouteNextHop
25	0.0284	Intrln042DF4+Gould	000026	Telnet C PORT=2355 rm de
26	0.1513	Intrln042DF4+Gould	000026	Telnet C PORT=2355 rm dea
27	0.0994	Sun	00394E+DEC	07F2F1 SNMP Got tcpConnRemAddress ... t

## DETAIL

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = interop

SNMP: Command = 0 (Get request)

SNMP: Request ID = 591479685

SNMP: Error status = 0 (No error)

SNMP: Error index = 0

SNMP:

Frame 19 of 188

Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

## 15-2 The Get Request

SUMMARY	Delta T	DST	SRC	
19	0.0053	DEC	07F2F1+Sun	01E954 SNMP Get ipRouteNextHop
20	0.1669	Intrln042DF4+Gould	000026	Telnet C PORT=2355 rm d
21	0.1128	Sun	00394E+DEC	07F2F1 SNMP Got tcpConnRemAddress ... t
22	0.0068	DEC	07F2F1+Sun	00394E SNMP Next tcpConnRemAddress ...
23	0.0465	Sun	01E954+DEC	07F2F1 SNMP Got No such name ipRouteNex
24	0.0048	DEC	07F2F1+Sun	01E954 SNMP Get ipRouteNextHop
25	0.0284	Intrln042DF4+Gould	000026	Telnet C PORT=2355 rm de
26	0.1513	Intrln042DF4+Gould	000026	Telnet C PORT=2355 rm dea
27	0.0994	Sun	00394E+DEC	07F2F1 SNMP Got tcpConnRemAddress ... t

## DETAIL

SNMP: ----- Simple Network Management Protocol -----

SNMP:

SNMP: Version = 0

SNMP: Community = interop

SNMP: Command = 2 (Get response)

SNMP: Request ID = 591479685

SNMP: Error status = 2 (No such name)

SNMP: Error index = 1

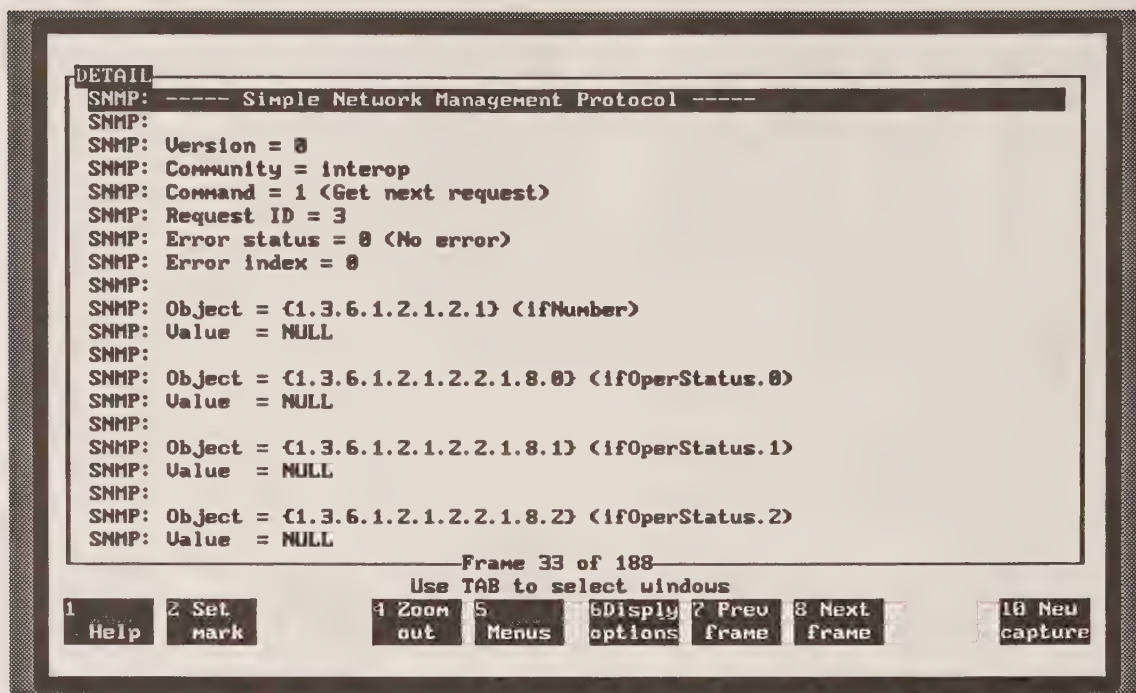
SNMP:

Frame 23 of 188

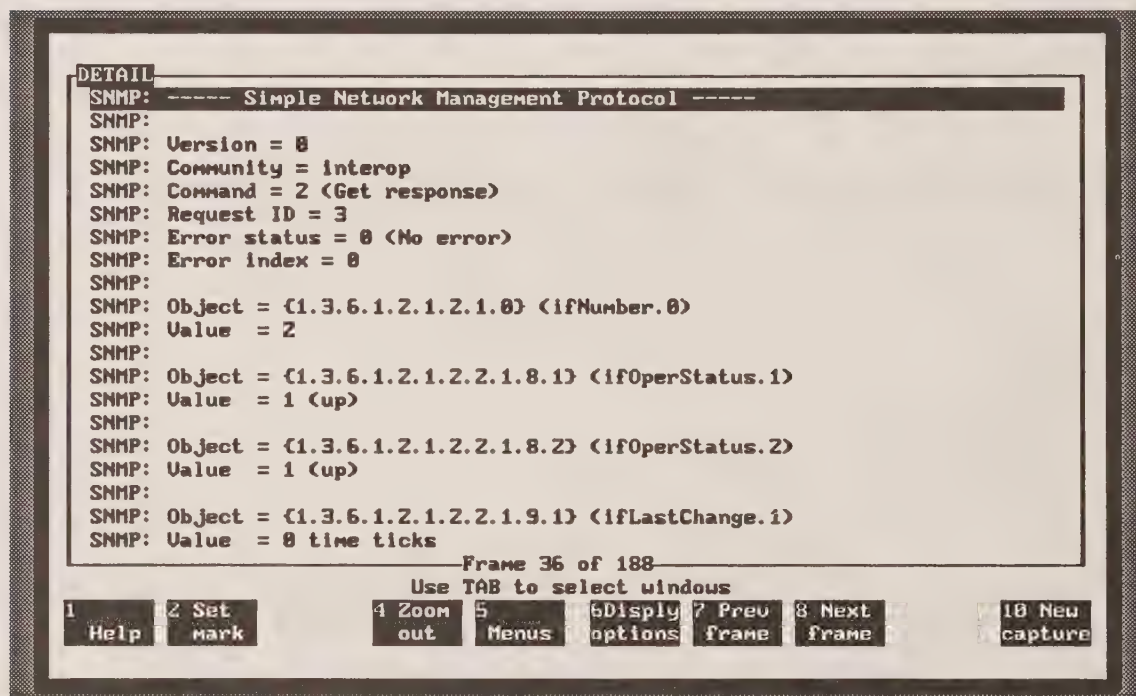
Use TAB to select windows

1 Help	2 Set mark	4 Zoom in	5 Menus	6 Display options	7 Prev frame	8 Next frame	10 New capture
--------	------------	-----------	---------	-------------------	--------------	--------------	----------------

## 15-3 An Error Response



15-4 A Get-Next Request



15-5 Response to the Get-Next Request

all the columns of a table. The `get-next` can thus be used to walk through the rows of a table. Figure 15-5 shows how each of the variables in the previous request has returned a new variable along with the value.

## SunNet Manager

SunNet Manager is a development platform as well as a management system. A series of modules come with the software, allowing management of objects accessible via a variety of different kinds of agents. Standard software modules allow the discovery of nodes on a network, management via SNMP, graphing of periodic data reports, and a variety of other functions.

In addition, the software contains a series of libraries and definitions that allow a user or software developer to add other kinds of modules. New MIBs can be incorporated, languages like NICE can be used to communicate with other agents, and presentation or functional modules can be added to process the information.

From the developer's point of view, the software has three basic APIs:

- Manager Services API
- Agent Services API
- OpenWindows API

OpenWindows is the library to access the user interface. Based on a combination of display PostScript and X Windows, the OpenWindows library allows the programmer to display information on the screen.

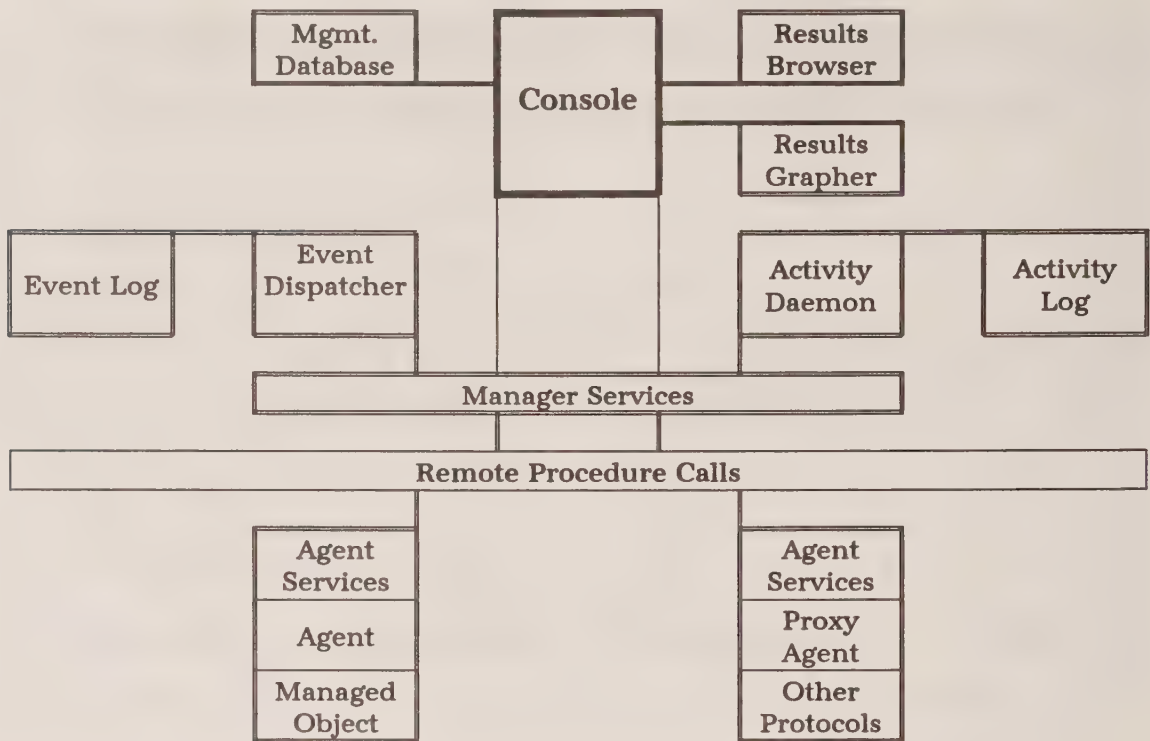
The Manager Services API is the other interface the application developer would see. This gives the programmer access to the SunNet Manager environment, allowing software to find and query objects on the network.

To add a new agent, the Agent Services API would be used. The Agent Services API means that the agent developer is unaware of the details of formulating RPC calls, opening sessions with managers, and other administrative tasks.

The SunNet Manager console runs on a workstation with the OpenWindows software. The workstation doesn't have to be dedicated to the task, although lots of requests can mean lots of data coming in, consuming resources at a fairly rapid rate.

The user interface on the console provides two basic sets of services. The display services are the link to the user's screen. In addition, there are selection services that allow the manager to find a particular network component and to initiate a desired function. With the selection services, the manager organizes the network as a hierarchy of component maps. Below the selection and display services are the request management services. These isolate the console element from changes in the underlying manager.





15-6 Components of SunNet Manager

Figure 15-6 shows a simplified view of the SunNet Manager architecture. The console is what the user works on. This console uses the Manager Services library to issue requests for certain types of data. The Manager Services library communicates that information using the RPC protocols to various agents on the network.

Some of these agents, proxy agents, will forward the data to agents that they manage, as in the case of the SNMP proxy agent. Other agents respond directly for a managed object and send the results back.

Once information is received at the SunNet Manager console, it can be displayed using the results grapher or any other modules that are written for a particular environment. Using the standard API, it is a fairly simple matter to add new agents or management modules.

### *Agents*

The agent library provides basic functions needed by all agents. It provides a mean by which the agent can send event, error, and monitoring reports. It also lets the agent initialize connections with various managers.



SNMP	SNMP Proxy agent.
PING	Reachability via ICMP echo, tests of round-trip packet time.
HOSTIF	Generic interface information (e.g., number of packets sent or received, errors, collisions).
SYNC	Synchronous interface agent.
X25	Information on virtual circuits, current link state information.
ETHERIF	Ethernet interface.
LAYERS	Information about TCP/IP protocol suite.
IPROUTES	Routing table information.
TRAFFIC	LAN traffic.
RPCNFS	RPC and NFS statistics.
HOSTPERF	Uses the rstat ONC service.
HOSTMEM	Host memory dedicated to network structures. On a Sun, this tells you about STREAMS and mbufs.
DISKINFO	Provides disk utilization information.
IPPATH	Provides information on connectivity between IP nodes.
LPSTAT	Provides printer status information.

### 15-7 SunNet Manager Agents

A wide variety of agents come standard with the basic SunNet Manager software. Figure 15-7 shows some of these basic agents. In addition, it is an almost trivial task to put together a new agent. The developer's toolkit includes sample agent source code, a tutorial, and a test manager to see if the new agent works properly.

Most agents in this environment communicate with the SunNet Manager console using the RPC and XDR protocols. This interface means that secure RPC authentication can be used to make sure that agents are only managed by the proper managers. The UDP transport protocol is used underneath RPC.

Some agents are proxy agents. They accept standard RPC calls in the SunNet Manager procedure library. Then, the proxy agent turns that request and sends it out via some other procedure. Many objects, for example, can be managed via SNMP. An SNMP proxy agent can be used to access native SNMP objects.

Proxy agents can also perform a valuable filtering function. A typical request to an agent is a request to be notified when a certain activity occurs, say a host crashes. To see if a host has crashed, somebody has to poll all the hosts periodically to see if they are operational.

If that polling function were done over a wide-area network, expensive bandwidth could be rapidly used up. Instead, the proxy agent performs

this function, and a wide-area packet only needs to be sent when an event of interest occurs.

### The Management Database

The SunNet Manager console uses a management database, which is a collection of structure and instance files that describe the network. Structure files (\*.schema) contain the object data structures. Instances of objects (and their relationships to other objects) are stored in instance files (\*.db).

In a distributed environment, we can make use of another RPC-based program: NFS. There is no reason why schema and database files can't be stored on a remote server and shared among multiple managers. Those files that are unique to one particular operation can be local mounts or can be limited-access NFS mounts. It is also possible to use SunNet Manager in conjunction with database management software.

### *SunNet Manager Elements*

The types of elements that can be managed are extensible. SunNet Manager has an elements.schema file that comes preloaded with the four basic types of elements. Although the categories of elements are fixed, elements within the categories can easily be added. The four categories of elements are:

- Components (e.g., printers, routers, workstations)
- Views (e.g., collections of elements, including other views)
- Buses (e.g., an Ethernet LAN segment)
- Connections (an element that connects two elements, such as an Ethernet, a leased line, or an RS-232 link)

Each element type is defined in the schema using a record. For example, the elements Sun-4, Subnetwork, and Toaster might be defined. A glyph (icon to use to represent the element) can be specified for any component or view, using the instance type of record.

As an aside, the toaster is used as the canonical SNMP object due to the efforts of John Romkey of Epilogue Technology. One of the architects of the INTEROP show network, Romkey once joked that he could put anything on the network, including a toaster. Dan Lynch, the President of Interop, Inc. dared Romkey to do so and the result was the first Internet Toaster. SNMP was used to access the MIB which would activate the toaster, which would in turn activate a PopTart.

It is even possible to define a set of user commands that apply to certain types of elements. When a user points to a glyph, a menu can be pulled down showing commands that apply to that element. Two default commands are rlogin and Telnet and others can be added (see Fig. 15-8).

```

record component.sun4 (
    string(32)      Name
    string(40)      IP_Address
    string(40)      User
    string(40)      Location
    string(80)      Description
    string(60)      _hidden_field
)
record view.subnet (
    string(32)      Name
    string(80)      Description
)
record component.toaster (
    string(32)      Name
    string(80)      Description
    string(20)      BrandName
)

instance elementGlyph (
    ( component.sun3      sun3.icon)
    ( component.sun4      sun4.icon)
    ( component.toaster    toaster.icon)
)

instance elementCommand (
    (component.sun4      "MyUtility"      "myutility %Name")
)

```

### 15-8 Element Definitions

An actual instance of an element is also stored in the database, represented textually as a cluster record. Cluster records are a means of collecting multiple records to make up a single instance. For example, Figure 15-9 shows a cluster record that applies to a specific Sun-4 element.

This cluster record shows that the Sun-4 is a member of a particular view and is connected to a specific bus. In addition, the agents available on that element are specified, as is the name of another server which can act as a proxy. The proxy means that if we want to query this particular Sun-4, we should send the request to the proxy agent running on SunEngServer, which will take responsibility for getting the information.

```
cluster (  
    component.sun4 (  
        SunEng14  
        1.1.2.288 Engineer23  
        "Building 2, Room 146"  
        "Sun-4/110 - monochrome, diskfull" )  
    membership ( Subnet2)  
    agent ( etherif )  
    agent ( hostif )  
    proxy ( hostperf SunEngServer )  
    agent ( layers )  
    proxy ( ping SunEngServer )  
    agent ( rpcnfs)  
    glyphColor ( 220 220 250 )  
    connect ( SunLAN)  
)
```

## 15-9 A Component Definition

### *Data Records*

Active requests are also shown as records in the database. A data request record would define the data that should be returned, how it is to be displayed, and where to put a glyph representing the request.

An event of interest differs across different managers. To some managers, it might be when a user uses too many resources. This would be the manager "beancounter." Another manager may be interested in security violations, and a third in system performance. To handle all these different types of needs, the SunNet Manager architecture incorporates the concepts of events and activities. An event is a report from the agent that a monitored attribute has changed. A user can specify that she is interested in reports on certain types of events. Second, an activity is a monitoring function that occurs repeatedly. Again, the user can specify that certain types of activity reports should be maintained.

Each console includes an event dispatcher service, which serves as the rendezvous for events reported by agents. This dispatcher moves event reports to the user and can make reports to a historical event log. The event log doesn't have to just include information from managed agents: programs on the manager can also put information in the log. For example, the SunNet Manager itself uses the log to report errors.

An alarm is a special type of event report. When an attribute changes in a specified manner, the user can request that the dispatcher take some special action, such as changing the color of an icon. For example, one user



might be interested in an alarm if a security violation occurs, and a second might be interested if CPU utilization reaches a certain point.

Activities are handled by the activity daemon. When an agent responds to an activity request for the first time, this activity is registered with the activity daemon in a log. Every 30 minutes, the daemon verifies that the requested function is still operating. If it is not, it is removed from the activity log. The user can then check the activity log to see which requests are active. If desired, the request can be automatically restarted after a failure.

### *Activity Requests in the Management Database*

Activities and data requests are all stored in the management database, allowing a standard series of requests to be shared between different users or different console systems. Figures 15-10 through 15-12 show an example of such a definition.

Figure 15-10 shows the beginning of a cluster that defines a data request record. The request has a name and can even be shown on the screen as an icon. How the event is actually displayed on the screen is a function of the second part of the definition cluster, the data attribute record (see Fig. 15-11).

Finally, Figure 15-12 shows an event request that allows a manager to register itself as being interested in certain types of event reports from certain types of systems. The accountant would register for “account overdrawn” events, the network manager would register for “illegal packets,” and the CEO would register for “nobody logged on in engineering.”

### *The Discover Tool*

To create a management database, the user can start with an existing one or use the Discover Tool. This tool runs in the background and automatically creates a graphical representation of the network.

As the Discover Tool finds hosts and networks, the user can use the console graphic editor to modify the graphical representation. It is also possible to create new objects directly inside the graphics editor.

The Discover Tool runs in superuser mode. When the user starts this function, a popup window starts up and asks for a root password. The tool begins by putting clouds into the main window: these each represent a network.

The user can then click onto a cloud glyph, which will show the hosts on that network. The hosts can be connected to a bus (an Ethernet) and moved around the screen to reflect their physical topology.

The way the Discover Tool works is to start with the routing table to find other nodes. It also watches incoming network traffic to find out about

cluster (	Every request, active or held, is defined by a cluster record.
dataRequest (	Beginning of a data report: the data request record.
hostperf.data.0	This is the name of the request. It is displayed under the request's glyph or in messages about the request.
0	Serial number of the request. Zero represents a new request.
SunEng14	Name of the target system.
SunEngServer	Name of the proxy system.
hostperf	Name of the agent.
data	Name of the group or table.
“Interval’ ‘Count’ Times”	The report time. Is either once, interval count times, or interval until canceled.
10	The interval field passed to the agent.
5	The count field passed to the agent.
“”	Options passed to the agent.
“”	Key field passed to the agent.
“”	File name for data reports. If null, reports are sent to the data log field.
undefined	An internal “fileport” variable for the console. This is set to undefined for new requests.
True	Boolean indicating if the request should be automatically restarted if it dies.
False	Boolean indicating if responses should be deferred until requested by the console.
“Save Request”	Value of either delete request or save request indicates disposition of request when it terminates.
undefined	An internal timestamp for the console. Set to undefined for new requests.
hostperf.data	The agent and group name (should match agent and group/table variable above).
SunEng14 )	The target object associated with the request. This is the name of an element in the database that is put into error messages

### 15-10 Data Request Record

other nodes. The tool will look up to ten hops away (meaning in a large network a very large amount of information can be returned into the manager database). Once a host is found, the Discover Tool looks for an SNMP agent or any of the other Sun-supplied agents.

With any discovery tool, there will be information that may not be automatically discoverable. For example, we may want to know who owns a

dataAttribute (	Zero or more data attribute records control how data is displayed by the console.
cpu%	Name of the attribute.
True	Boolean indicating if data should be displayed in the data log.
1	Internal console variable indicating the ASCII port.
True	Boolean indicating if this attribute should be displayed as an indicator.
1473696	Internal variable for the console indicating the bar graph port.
None	Indicates if the attribute should be displayed in a strip chart. Valid values are none, absolute values, or delta values.
0	The internal variable that indicates the strip chart port.
"Delta Values"	How attribute should be displayed in the grapher. Values are none, absolute values, or delta values.
1515136	Grapher handler (internal console variable).
)	
rqstState (	The request state record.
Idle	State of the request. Field should be active or idle.
0	Internal variable always set to zero.
0	Internal variable always set to zero.
)	
membership (	Final record of the data request. Indicates in which object's subview the request glyph should appear.
SunEng14	Name of the object. Normally the same as the final value of the data request record.
)	End of membership record
)	End of the data request cluster.

## 15-11 Data Attribute Record

particular node, who is responsible for managing it, and other information such as the physical location of the object.

### SNMP Support

SNMP support is provided on a Sun in the form of a proxy agent, allowing management of remote SNMP devices such as routers. The Sun workstation itself is managed using other agents. The SNMP proxy agent (na.snmp)

cluster (	
eventRequest (	
hostperf.data.1	Name of request.
1	Serial number of request.
SunEng14	Target system.
SunEngServer	Proxy system.
hostperf	Name of agent.
data	Name of group or table.
Once	Report frequency.
0	Interval field.
1	Count field.
""	Options field.
""	Key field.
False	Restart boolean.
False	Should request terminate after first report?
False	Should responses be deferred until requested?
"Delete Request"	Action upon completion.
undefined	Internal timestamp variable.
hostperf.data	Agent and group ID.
SunEng14 )	Target object.
eventAttribute (	
cpu%	Name of attribute.
"Greater Than"	Relation for first threshold.
80	Value for first threshold.
"Threshold Not Set"	Relation for second threshold.
""	Value for second threshold.
"Blink Glyph"	Signal options.
""	Name of file or username for a signal.
Low )	Priority (low, medium, or high).
rqstState (	
Idle	State of request.
0	Internal timeout value.
0 )	Internal request flags.
glyphState (	
32 )	State of the glyph.
membership (	
SunEng14	Name of subview the request should appear.
))	

### 15-12 An Event Request Cluster

uses RPC to communicate with the SunNet Manager console. From there, the proxy uses SNMP to talk to the agent.

A manager can use both standard SNMP MIB objects and enterprise-specific ones. When the proxy agent receives a request for a particular SNMP agent on a device, it searches the /var/adm/snm/snmp.hosts file for the



SNMP schema file to use for that device. If it can't find an entry for that device, it uses the default schema.

The standard default schema is MIB II (or MIB I if you want). If you look in `/etc/snm.conf` under the keyword `na.snmp.default-schema`, you'll find the location of the default schema. An organization can create its own SNMP schemas or use ones from vendors.

There is a `mib2schema` script that converts a MIB into a SunNet Manager schema. Chances are that after using `mib2schema`, you're going to want to improve the description text, add formatting information to the characteristics field, and if necessary, modify the array sizes of stringtypes.

There is a special proxy agent called `na.snmp-trap` which handles asynchronous or unexpected reports. By default, this proxy then sends trap reports to the event dispatcher on the local machine. The proxy can be directed to send trap reports to multiple rendezvous points.

There is an SNMP trap file which holds enterprise-specific traps. It consists of a series of lists. The list begins with an ID (which begins with the keyword `enterprise`) followed by your object identifier. The object identifier is followed by trap numbers and names:

```
# Sun Microsystems
enterprise 1.3.6.1.4.1.42
  0      sun-specific-0
  1      sun-specific-1
```

### *Network Management Security*

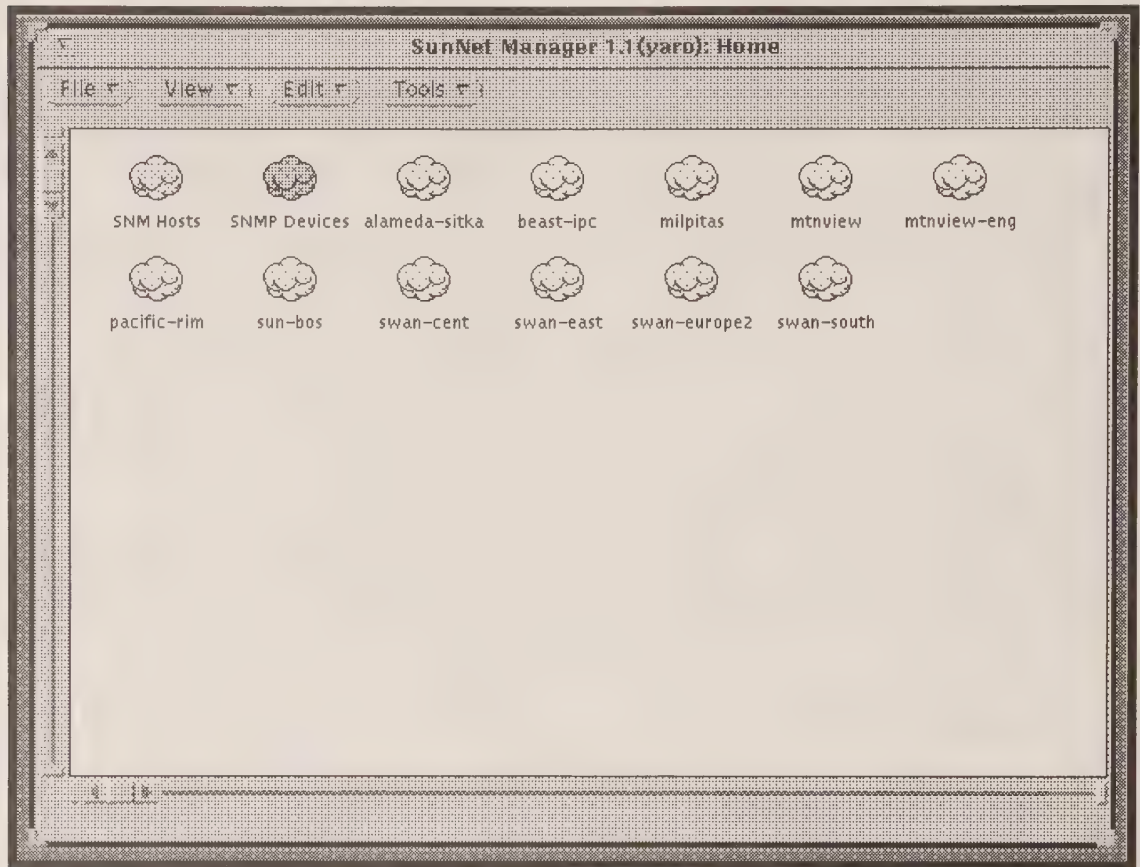
There is an optional security mechanism to restrict access to agent services. As delivered, you can ask for information from any agent. For each agent, the manager can specify read-security for data and event report requests and write security for set requests.

Security is built on top of Secure RPC. Once it knows who the user is, SunNet Manager checks access rights based on `netgroups`. Access rights are administered by making people part of one of five different network security groups.

The agent library has a global variable, the network management security level, that has a value from 0 (not secure) to most secure of 5. Setting this variable determines which class of users will be allowed to use an agent.

An agent goes through six steps in deciding whether to grant a request when security is enabled. First, if NIS is not running, the request is rejected. Next, the agent looks at its own security level. If the level is zero, all requests are allowed.

Next, the agent verifies that the user has a Secure RPC authenticator. If so, it checks the domain-specific user ID and password table for the network name supplied in the Secure RPC authenticator. Once membership is con-



15-13 A View of Views in SunNet Manager

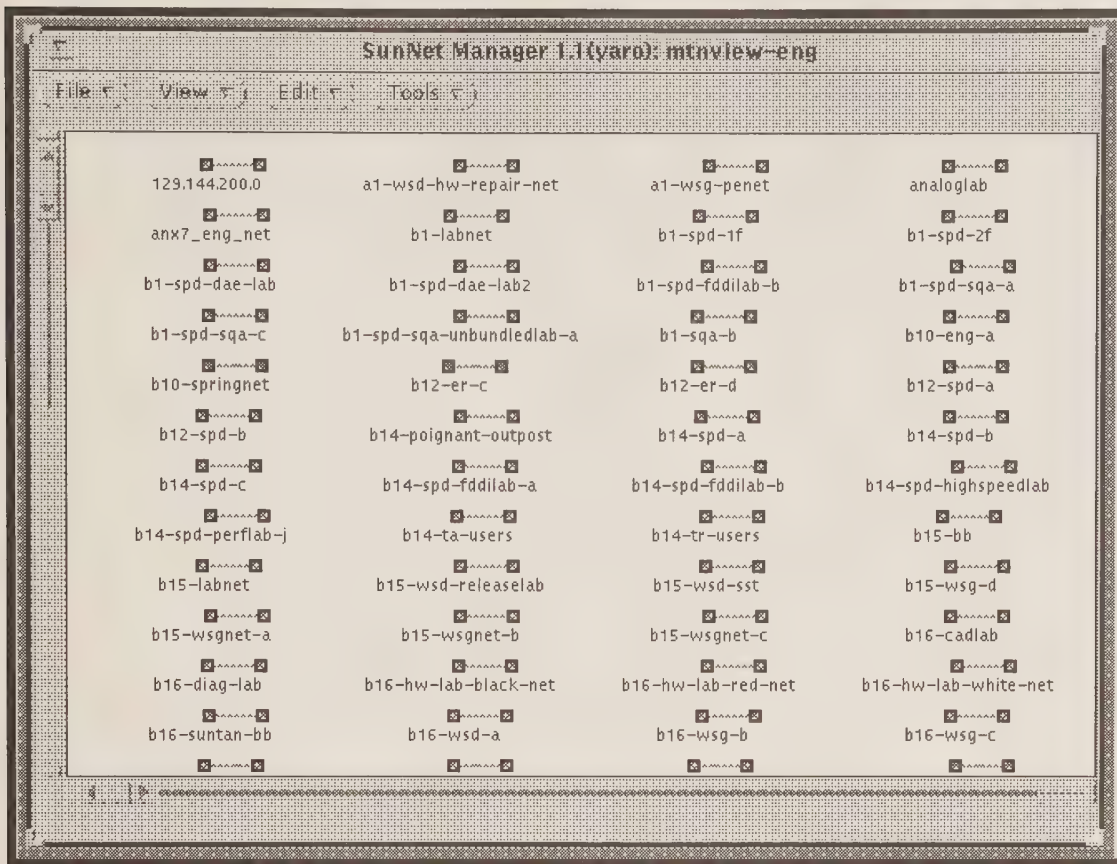
firmed, the agent checks the security level and decides whether or not to grant the request.

### Operation of SunNet Manager

The objects in a network are grouped into views, shown as clouds on the console. Those clouds can in turn be part of larger clouds. A network finally resolves to a top-level map, usually simply a series of clouds (see Fig. 15-13).

The basic operation is to choose a top-level map of the network. Clicking on one view yields the contents of that view. At some point, actual objects are reached. The typical objects are Sun workstations, other computers, and, of course, connections between the systems.

All this information is stored in the management database. Filling the management database with information is either done as a configuration



Courtesy of Sun Microsystems

15-14 A Series of Networks Discovered

option loaded in a text file or through the Discovery Tool. Figure 15-14 shows a series of networks discovered using the Discover Tool.

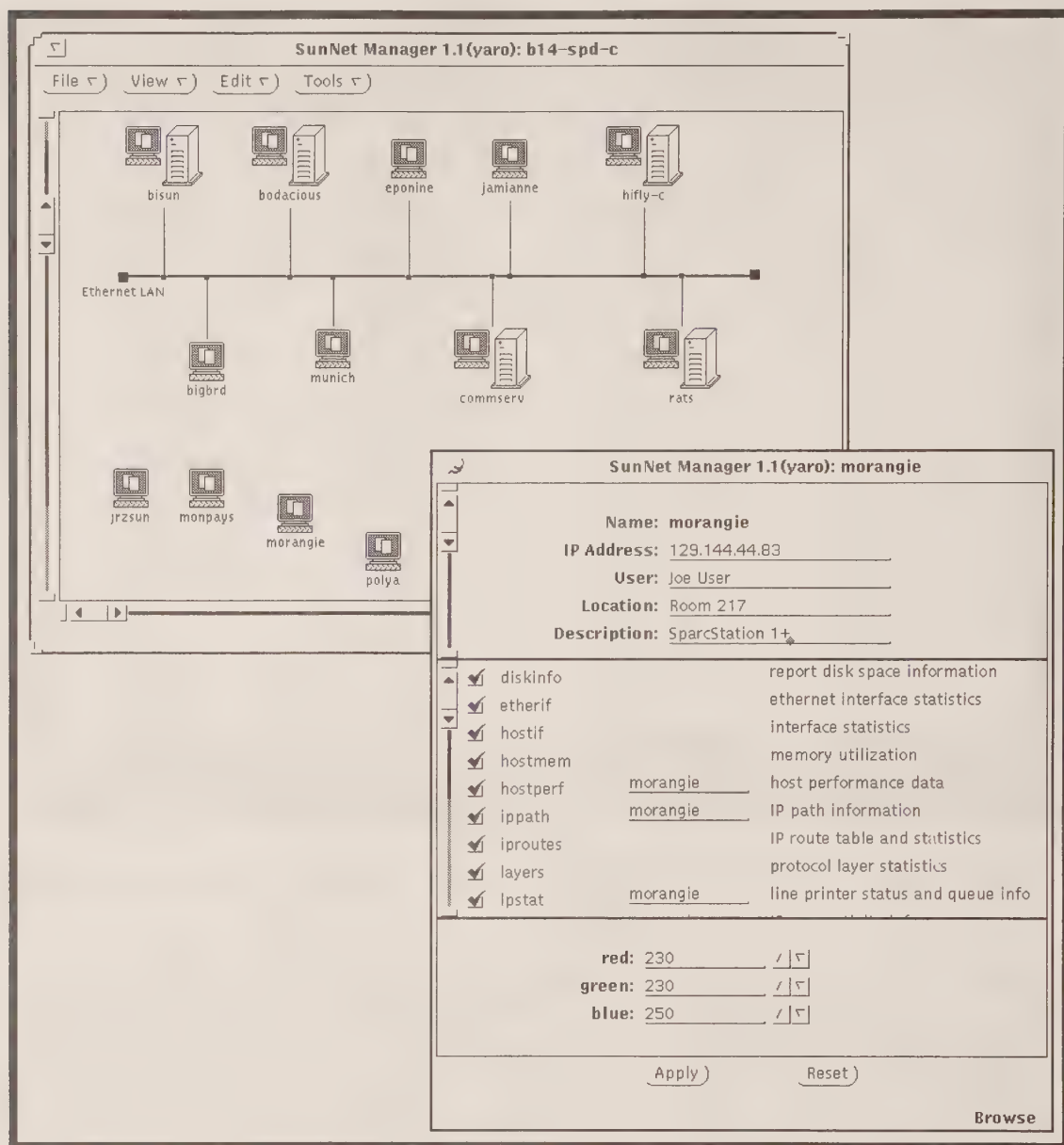
After discovering a network, a user can click on it and see a screen that contains a glyph for each system found. Those glyphs can be connected together onto a bus or simply left floating about (see Fig. 15-15).

Whether or not the systems are connected together with a line is immaterial to the operation of SunNet Manager. If the system is to be monitored for a long time, it makes sense to make pretty pictures. The graphical placement of the objects can also be used to add new, unmanaged objects, such as printers or modems that do not have SNMP support.

Each system can be defined in terms of the information it can provide to a user. As shown in Figure 15-15, the host Morangie definition form is on the screen. The user can change the color of the icon and can indicate which agents are to be used for this host.

In addition to systems, it is also possible to define activity or event requests and map those to the screen. Figure 15-16 shows an example of an



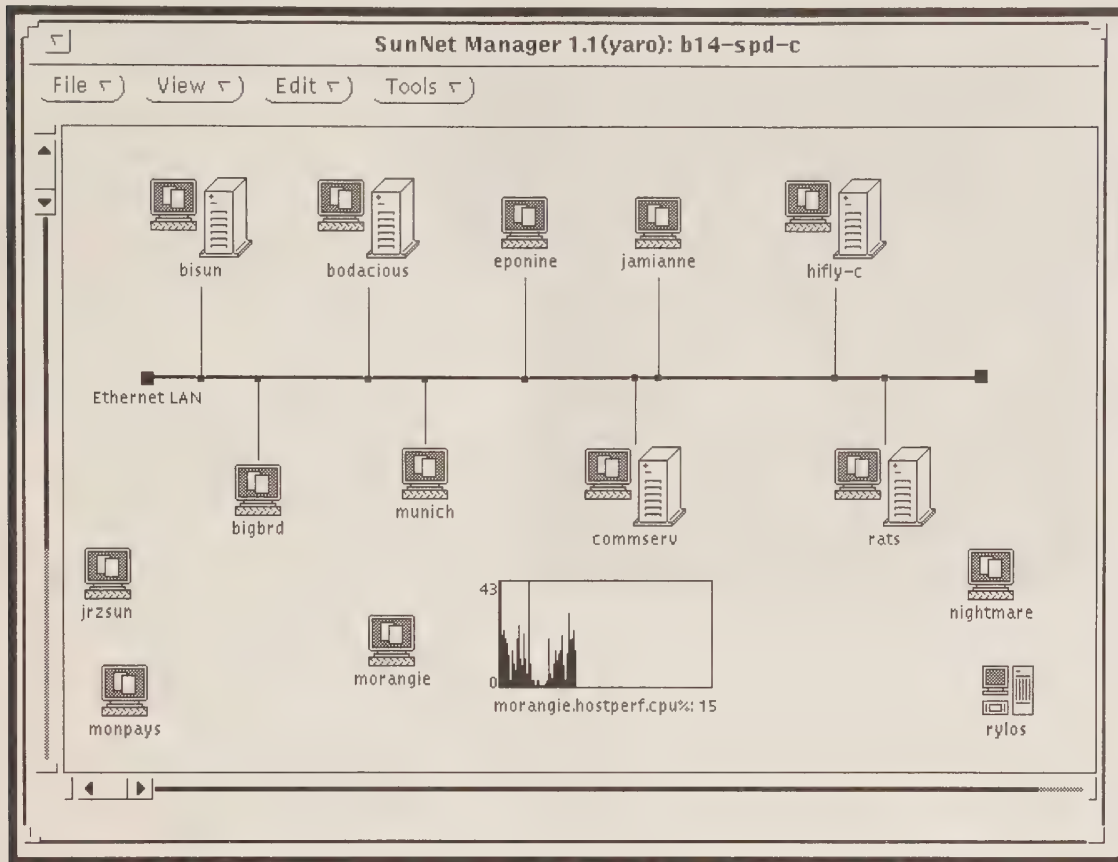


### 15-15 Defining Morangie

activity request for the CPU utilization of host Morangie being displayed as a strip-chart on the screen.

Events can also be displayed through a variety of different means. A common one is to have an action occur to the icon on the screen when an event occurs. For example, if average CPU utilization reached 60 percent the icon could become orange; over 90 percent, the icon could go red. Missing machines could be displayed in black.





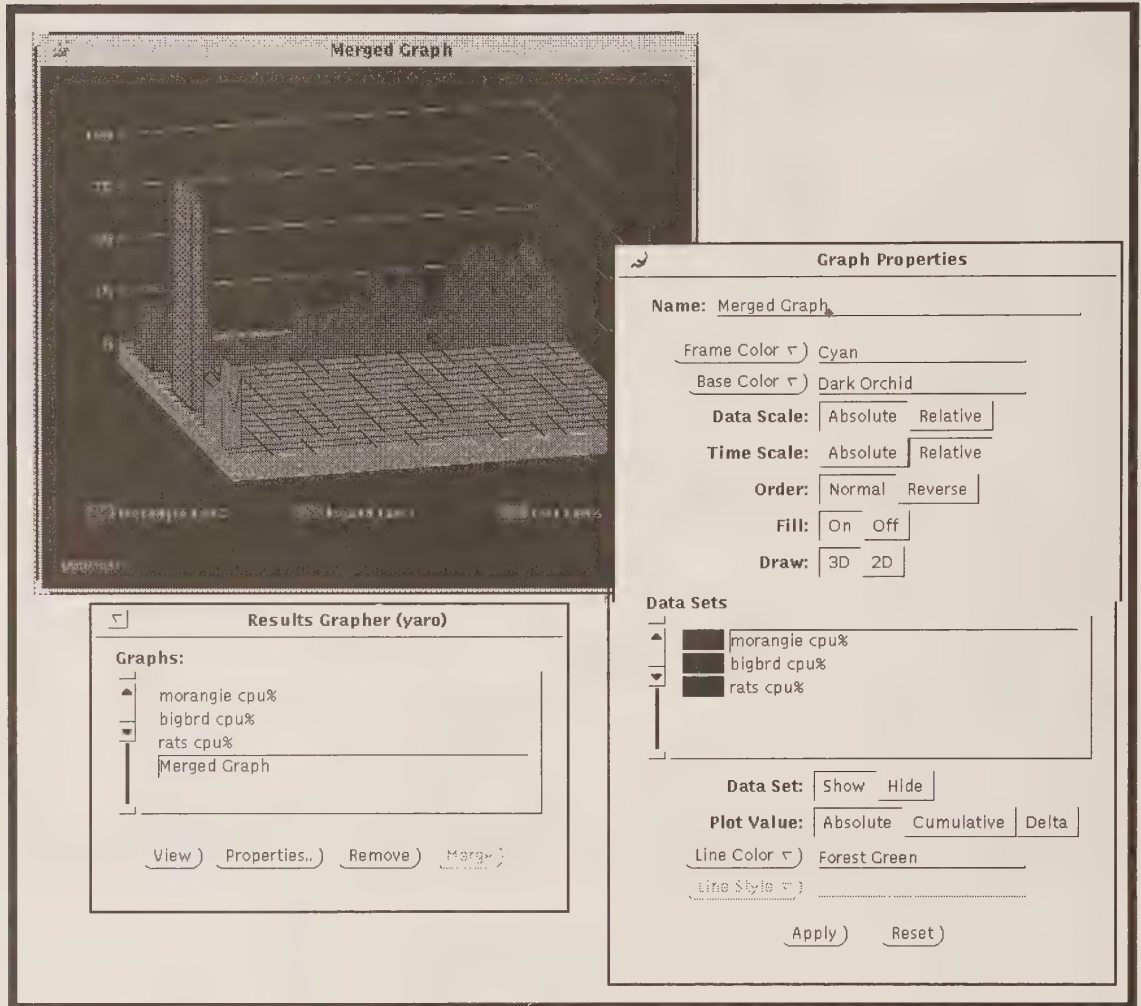
**15-16** Graphical Display of a Data Request

It is also possible to combine several of the data requests together for a unified graphical view of the network. Figure 15-17 shows an example of a merged graph. Notice that there were three CPU utilization reports defined, each represented as a data set in SunNet Manager.

The Graph Tool allows the user to define a merged graph in which several data sets are combined. Notice that the graphs for two of the hosts (bigbird and rats) were just started, so the merged graph has more data for the host Morangie than for the other two. Notice also that it is possible to use a variety of designer colors, represented as dithered dots on the black and white dumps used in this book. The colors allow the network manager to use visual clues to find significant events on the network.

## Basic Management Tools

Most of the agents in SunNet Manager are really a unified means of accessing common tools. For example, before the days of fancy consoles, many



15-17 Merging Several Data Requests

managers used the ping command. This simply uses ICMP to test the reachability of a node on the network.

The Ping agent is a fancier way of using the ping command. Rather than executing the command manually, activity reports can be issued that specify periodic pinging of a set of hosts, recording the results in a database. This is not to say that the original, command-driven, ping command is useless. When a user calls up the manager on the phone complaining that a server is down, most managers will move their mouse over to a console window and type “ping servername” to see if the reputed host has crashed (or doesn’t exist):

```
% ping elvis
elvis is alive
```

```
% ping elvis  
no answer from elvis
```

A second basic command is `ifconfig`, used to configure network interfaces and to get information back. A manager can test not only if an interface is working, but which parameters it is using. In the case of an Ethernet card, for example, we would probably want to know if it is accepting broadcasts. If it isn't, that might explain why it wouldn't respond to broadcast requests such as a name server bind.

### For Further Reading

Rose, Marshall, *The Simple Book*. Englewood Cliffs, N.J.: Prentice-Hall, 1990. A good introduction to SNMP and SMI.

M. Rose, K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC 1155, May, 1990.

Sun Microsystems, *SunNet Manager 1.1 Installation and User's Guide*, Part No: 800-5512-10, Revision A of 1 February 1991.

———, *SunNet Manager 1.1 Tutorial—How to Write an Agent*, Part No: 800-5514-10, Revision A of 1 February 1991.

———, *SunNet Manager 1.1 Tutorial—How to Write an Manager*, Part No: 800-5513-10, Revision A of 1 February 1991.

V. Cerf, *Report of the Second Ad Hoc Network Management Review Group*, RFC 1109, August, 1989





# Interoperability



# Interoperability

## Overview

So you've done your job—your network may not be perfect, but most of the problems get solved one step ahead of the users. If your users started reading more manuals, they might start asking for more than you can deliver but, human nature being what it is, things seem to have reached a steady state. Steady state until a manager calls you in and places you in charge of two networks instead of one. Needless to say, the two networks can be run as separate islands of automation, but to do so would run against the nature of most MIS managers.

After numerous meetings, you finally find yourself in the position of having to integrate two entirely different networks. Most large organizations being what they are, the networks will, of course, use different architectures. This chapter looks at how these different computing environments can be brought together. The simplest way is to make them all speak TCP/IP and NFS. All computing environments, with only a very few minor exceptions, support both these sets of protocols.

If the TCP/IP route is chosen, it is simply a matter of picking the appropriate vendors for each of the platforms and then choosing some common subnetwork media to tie it all together. We assume that everybody read the standards the same way so the systems work together. This last assumption may make some cringe. Both the TCP/IP and the ONC stacks have been around for long enough that the assumption is valid, at least for the core subset of functionality. There are certainly bells and whistles (Telnet negotiations come readily to mind) that don't work as advertised, but the basic TCP/IP and NFS services do work very well across different platforms.

There are times when a single architecture is not desirable technically or possible politically. Luckily, there are only a few cases where this poses a real problem. This chapter starts by looking at what is certainly the biggest challenge, integrating IBM's System Network Architecture (SNA) with Sun

TCP/IP and NFS networks. After we have looked at SNA interoperability, we will go back and look at the rest of the world, starting with DECnet. We will see that it is a fairly simple matter to integrate Sun equipment into DECnet or DEC equipment into TCP/IP.

Next is the question of the PC. The PC, lacking the rudimentary services in the operating system we would expect of any decent system, poses a larger challenge for integration. Luckily, most PC network vendors are beginning to realize that the PC network does not stand alone and are providing hooks for integration into a TCP/IP environment.

This book concludes with a few words about TCP and OSI interoperability. We have seen OSI protocols in use throughout this book. The two protocol stacks are both available on Sun-based environments, and there are a variety of gateway services available.

### The Special Case: SNA

The approach to IBM integration is twofold. An easy solution is simply to make the IBM system speak TCP/IP and NFS. We will return to this approach later. For now, let us consider the administratively more realistic case of keeping the mainframe systems as an SNA-based network.

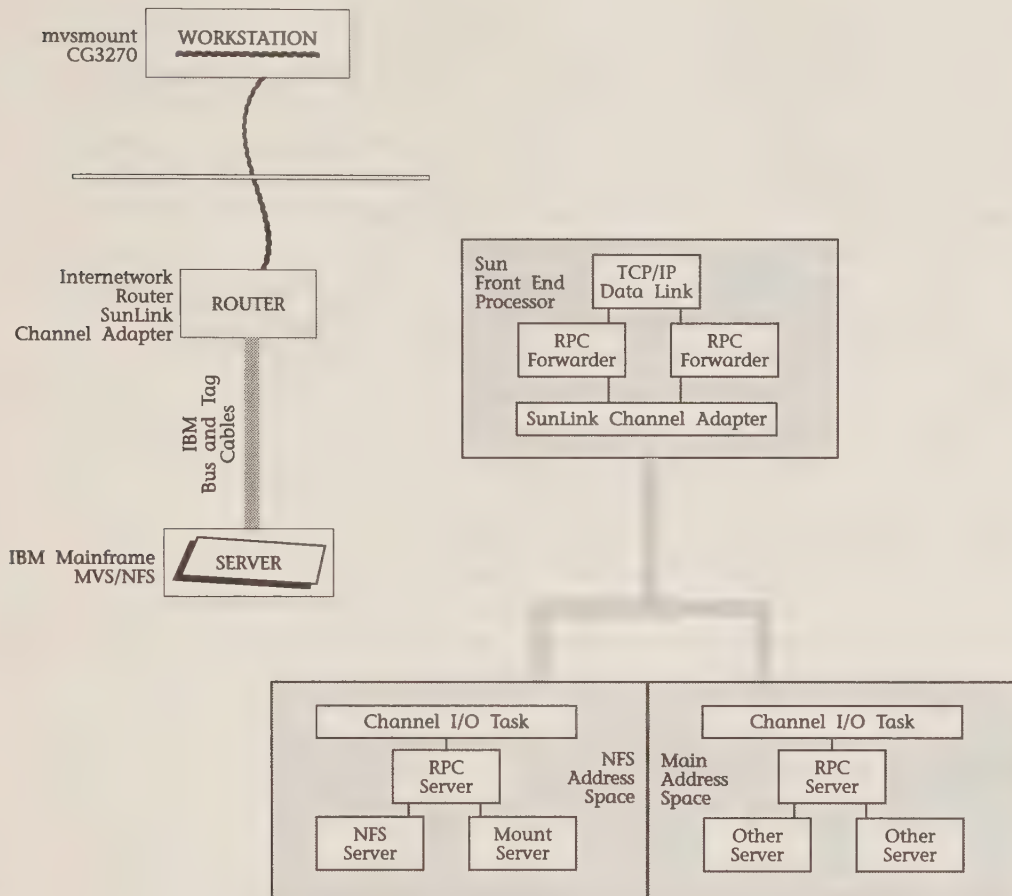
Connecting the two environments together requires a gateway system. Gateway systems can be run in a variety of different configurations. A single user going to a single host using software like Remote Job Entry (RJE) can easily run on any Sun with a local serial port. Connections to more hosts with more users or more data can use a SunNet Multiprotocol Communication Processor (MCP), a multiprotocol synchronous board which supports the Bisynch (BSC) or SDLC data link protocols. Even higher performance interconnection can be provided by direct attachment to a channel on a mainframe.

It should be remembered that communications, like any function, take CPU resources. Even going out of a local serial port at low line speeds can take 5 percent of a Sun-3's CPU for a constant data stream. Higher speeds take more of the CPU—a single 38.4-kbps line can use 40 percent of a Sun-3.

### *SunLink Channel Gateway*

The fastest way of communicating between the two environments is the Channel Gateway. The IBM channel is a very high-speed interface, often used for connecting to other hosts, to communications controllers, or to other high-speed peripherals (see Fig. 16-1). The Channel Gateway is a double VME card slot board assembly, a set of bus and tag interface cables, and Channel Supervisor software. Together, these let you connect to the IBM mainframe block multiplexer channel.





## 16-1 Sun Channel Gateway

A single IBM channel has addressing space for up to 256 different devices. Up to 8 physically separate control units or 16 logically separate control units can be connected to a channel. A Sun server acting as a gateway can simulate one or more of these control units (up to the maximum). If you support multiple control units, they can be the same or different control unit types.

For each control unit, an application called a control-unit emulation is run. A typical control-unit emulation would be SunLink Local 3270. Each control unit emulation has a definition file and an emulation program. The definition file has the channel commands that the control unit uses and groups commands into classes based on their functionality.

Version 7.0 of the Channel Gateway has an exchange feature. This is an emulation of the IBM Channel-to-Channel adapter which allows you to run

the IBM NJE protocol over the Channel Gateway to do bulk file transfer, job submission, and interactive user messages.

Channel Gateways can coexist with other devices on the same channel (including other Channel Gateways). It runs on the Sun-3, Sun-4, and SPARCstation (with four or more 9U slots, a local disk, and at least 4 Mbytes of memory). Multiple control units require more memory.

The Channel Gateway has a maximum bandwidth of 3 Mbytes/second. The NJE bottleneck is the fact that incoming files are spooled and then queued for outgoing transmission. Fastest performance will be with a fast CPU (e.g., a Sun-4) and a fast disk (e.g., an SMD-based drive).

Figure 16-2 shows a sample configuration to connect Sun (and any other TCP/IP) clients to the SNA environment. The configuration was an actual test system set up at Sun. Notice that a variety of workstations are in use, all running TCP/IP and NFS. There are two large IBM systems: an IBM 4341 and an Amdahl 580, both running MVS/NFS.

Notice that the Channel Gateway system connected to the Amdahl is also an internetwork router. The High-Speed Serial Interface (HSI) is used to connect the combination gateway/router system to another router, using a 15-mile T1 line. Any of the workstations pictured can do mounts of any of the MVS-based servers.

### *SunLink 3270 Products*

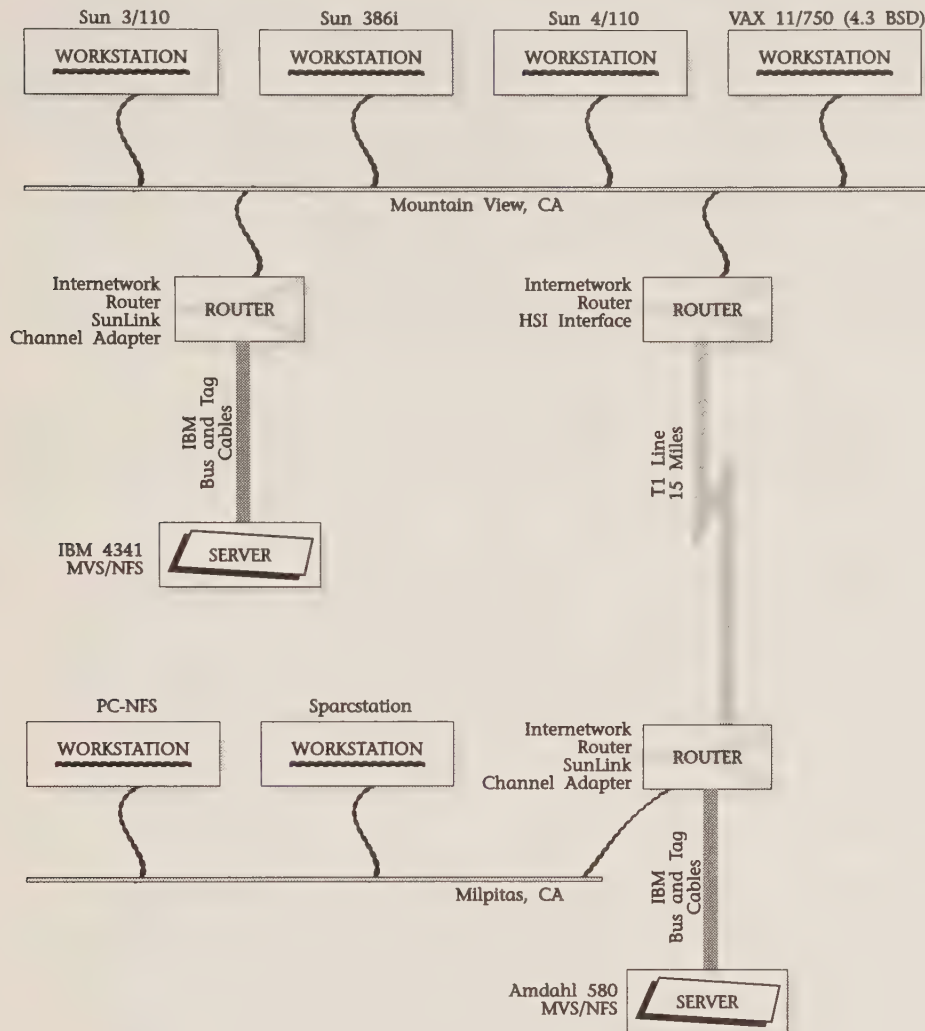
Before getting back to some sample configurations, we will first look at some of the software services available over gateways. MVS/NFS was discussed in Chapter 6. The other basic service is emulation of a 3270 terminal. There are three different ways to emulate a 3270 terminal:

- SunLink Local 3270
- SunLink SNA 3270
- SunLink BSC 3270

The local configuration runs on the Channel Gateway. SNA and BSC are for gateway workstations (not Channel Gateways) that use either SNA or BSC to get to the mainframe. The basic idea is to emulate a 3274 cluster controller at the gateway and then support a 3278 terminal on the workstation, as well as an IBM 3287 printer someplace. The virtual terminal and printer can be running anywhere on the TCP/IP-based internetwork.

In addition, programmers have available a 3270 datastream application interface and the IBM 3270 PC-compatible file transfer mode, which works on both the VM and MVS operating systems.

A variant is TN3270, for Telnet 3270. This allows 3270 terminal emulation on top of the TCP/IP Telnet protocol. TCP/IP must be running on the host and an Ethernet Attachment box must also be attached to the mainframe. TN3270 allows a valuable alternative to a Channel Gateway, making



16-2 Sample Integrated Network

the mainframe speak TCP/IP but still supporting 3270 emulation. Even if TN3270 is being used, there is still a role for a gateway system. Background file transfers and peer-to-peer programs still need to interact with the mainframe using the SNA protocols.

### *SunLink CG3270*

The SunLink CG3270 software allows emulation of the IBM color graphics terminals. Technically, the software emulates the IBM 3179G display terminal. The software runs on top of the NeWS window system.

The software is an order-driven device, meaning that the virtual terminal is sent a set of picture component descriptions: curves, vectors, text, and areas. The workstation translates the components into the PostScript language and puts them into a NeWS window. One nice feature of being inside a PostScript-based windowing system is that the user has full control of sizing. The window can be dynamically resized and the contents inside the window will automatically scale. Graphics objects inside a window can be captured, moved to another window, and locally manipulated.

Several utility windows are available that give the user control over the information in the terminal emulation window. A color mapper allows mapping of the 8 colors on a 3270 terminal into the 16.7 million colors on the Sun workstation. A keyboard mapper allows the mapping of an IBM function or keystroke to a Sun keyboard key. A file transfer utility uses an IBM program called the 3270-PC File Transfer Program. A third utility is an output control window which allows screen dumps to be placed into files, a printer, or any other device.

The CG3270 software is built on top of the Datastream Access Interface (DAI) programming interface. You need one of the SunLink gateway products available (SunLink SNA 3270, SunLink BSC 3270, or SunLink Local 3270). Licensing for CG3270 is built on top of the SunNet License Server. Users pay for the software based on the number of simultaneous CG3270 sessions they would like.

### *SunLink BSC RJE*

Bisynch (BSC) is a name for a very old IBM network, long ago supplanted by SNA. Bisynch is so old, in fact, that it really just defines two layers: physical and data link. Built on top of Bisynch are a few primitive services, such as a Remote Job Entry facility. So why would Sun implement support for BSC/RJE? For that matter, why are they mentioned in this book?

BSC/RJE is proof of the longevity of technology. It still provides a lowest common denominator interface to several mainframe systems. There are three services that are of interest here: 2780, 3780, and HASP. All of these work on both the MVS and VM operating systems. The basic service is remote job entry using a card deck. The Sun gateway emulates a card reader, complete with a "deck" of 80 character records containing the appropriate control, program, and data cards. In addition to emulating the card reader, the software emulates a card punch, a printer, and a console.

The gateway is responsible for job queuing and job submission to the host. The gateway can have multiple jobs queued, using a first-in first-out method. Since the gateway is a Sun workstation, the files that make up the job can be located anywhere on the network.

Jobs can be received in several different ways. One option is to have the RJE gateway put all jobs in a single directory. Alternately, jobs can be sent



to a specific user's directory. A more sophisticated option is to put a comment card in the output, which invokes a SunOS shell command to do further processing. Again, the reader may ask why one would bother doing this. Think about an environment which has an old, very large application: say the corporate order processing system.

Often, some of the functions that used to be performed by hand-punching card decks can be automated. If all the users have a Sun workstation (or any other workstation or PC), data entry can be simplified considerably. Windows, forms, data validation, and context-sensitive help can all be added—all features, we might add, that are missing from your average card reader.

The mainframe program does not need to be changed at all. It still operates the way it always did, reading or punching decks of cards. The only difference is that the deck of cards is automatically generated by the workstations and submitted over the gateway system.

### *SunLink SNA Peer-to-Peer*

The SunLink SNA Peer-to-Peer software is considerably more advanced than either the BSC/RJE or the 3270 modes of operation. This software is based on the IBM LU6.2 and Advanced Program-to-Program Communication (APPC) standards which encourage a less centralized form of network interaction by facilitating peer-to-peer communications. The SunLink software allows the programmer to emulate a Physical Unit type 2.1, the same node type used on token ring-based SNA networks.

Supporting APPC means that the programmer can interact with a program running on the mainframe, or any other peer system in an SNA network such as IBM's mid-size systems. The peer might be running commercial software. Alternatively, new programs can be written that make use of both the TCP/IP and SNA resources. In addition, the software supports the Document Interchange Architecture (DIA), Document Distribution Services (DDS), and Document Library Services (DLS). Support of DIA DDS/DLS means that the programmer is able to use the DISOSS library system on IBM mainframes.

A typical use of DISOSS is some corporation-wide information file, such as a corporate correspondence log. An insurance company might use DISOSS to store policies, for example. By interacting with the DISOSS services, the Sun programmer can integrate some more advanced user interface, such as Interleaf or Frame document processing to the corporate information store.

Interacting with DISOSS is just one example of how one might use the SunLink SNA Peer-to-Peer API. Perhaps a more typical example would be for custom in-house systems that take advantage of APPC to set up distributed transactions processing systems.

## SNA Case Studies

To see how we can connect the TCP/IP world to the SNA world, we will look at three case studies, each illustrating a different approach. The first case study is used at Sun Microsystems, and it uses a SunLink Channel Adapter. The second approach is one chosen by the Town and Country Building Society, a British banking firm. In that approach, the Sun equipment emulates an existing MVS-based program to smooth the transition from one environment to another. The third approach is quite simple: make the IBM speak TCP/IP and retrain the dozen or so SNA specialists.

Figure 16-3 shows the architecture used at Sun Microsystems to provide IBM connectivity. Three large IBM Plug-Compatible Mainframes (PCM) are used as servers, NAS AS/XL-50 and Amdahl 5890 mainframes. These computers run traditional applications: MRP systems for manufacturing, accounting, and personnel. The database management system is Cullinet's IDMS.

Connectivity is strictly through channel attached gateways. There is no physical SNA network, no front-end communications processors, and no cluster controllers. There are a few 3270 terminals, but only for systems programmers to use as consoles.

All other access is through a Sun Internet Router with a SunLink Channel Adapter board, known as an IRCA. A total of eight of these IRCA systems are attached to a single subnetwork in the Milpitas facility.

The IRCA systems are isolated from the rest of the corporate network by two routers. The rest of the EBay domain comes to the mainframes through these two routers and then on to one of the IRCA systems.

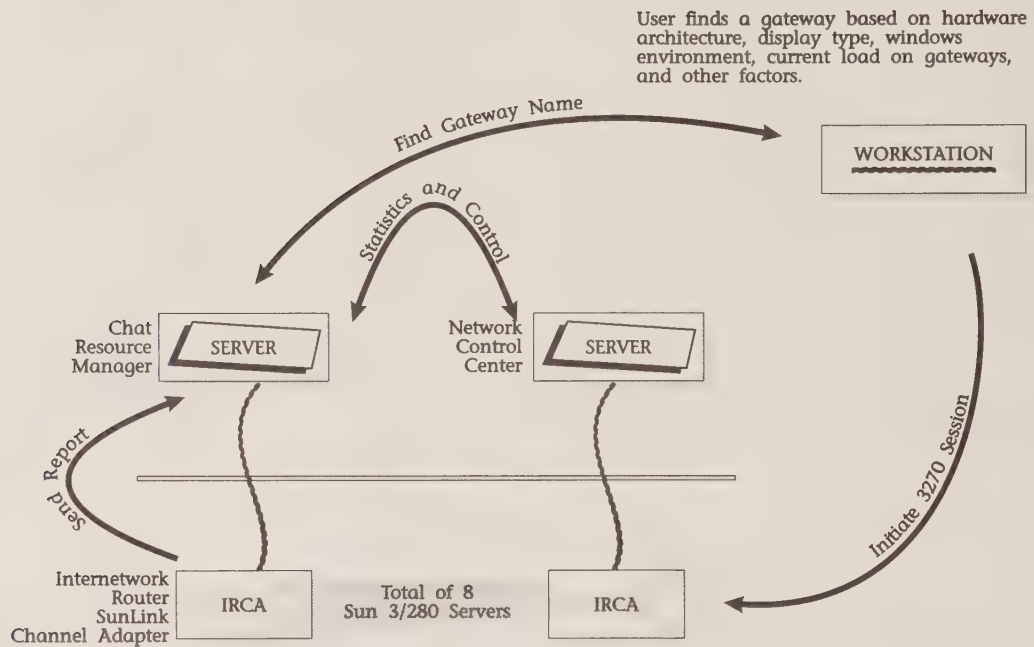
Notice that there is an additional IRCA system which is isolated on its own little subnetwork used for access from outside the Milpitas area. These users include corporate types, salespeople, and other users on the Sun Wide-Area Network (SWAN).

Because the outside traffic all comes in through this special SWAN subnetwork, it all goes through the same IRCA system, which in turn has its own channel connection to the mainframe. This partitioning of access allows greater control over security.

There are a variety of different applications used, but one of the biggest uses of the system is emulation of a color 3270 terminal using the CG3270 software to access interactive mainframe applications. Since users don't have 3270 terminals, they are able to use their workstation screens, a more pleasant method of interacting with MVS than the typical 3270 terminal.

In order to spread out access from different users across the different IRCA gateways, the corporate Information Resources staff wrote a simple little RPC-based program called SunAccess. Each IRCA has a capacity for a certain number of simultaneous sessions for different kinds of terminal

SunAccess matches the capabilities of a workstation with an available gateway system to provide load balancing. The basic operation of this program is shown in Figure 16-4. When a user wants to start up a session, the SunAccess command on the workstation contacts a load balancing server, known as the Chat Resources Manager.



#### 16-4 Load Balancing on Multiple IRCAs

other 3270 terminal) emulation package and starts up a session with the gateway.

As a point of interest, the CG3270 software, the most popular with users (at least those with a color workstation) also makes use of the License Server as a way of maintaining resources for certain groups of people. The license server handles the question of authorization and maximum concurrent users for software; the resources manager picks the IRCA that will provide the best performance.

This load balancing system is quite effective. Figure 16-5 shows a portion of the load balancing report for the system (there are at least 9 IRCAs in operation). Notice that the entire bank of IRCA systems was running at roughly 75 percent of capacity at peak times. In addition to the various MVS/NFS and printing tasks, this bank of IRCAs can handle roughly 1000 daily 3270 users.

What is particularly informative is the detailed usage section at the end of the report for each IRCA. Notice that the peak times for each of the IRCA systems were achieved at almost the same time and the same numbers. For example, IRCA 2 reached a peak of 146 Model 2 emulations at 13:26, while IRCA 3 reached its peak of 147 at 13:32.



Wed Jan 23 23:57:03 PST 1991

Percentage of IRCA Utilization for 01/23/91

Maximum usage = 1014

Limit = 1344

	0	10	20	30	40	50	60	70	80	90	100
HR+											
00+##		3									
01+##		4									
02+##		2									
03+##		1									
04+##		4									
05+#####	9										
06+##17#####											
07+##33#####											
08+##54#####											
09+##68#####											
10+##73#####											
11+##73#####											
12+##73#####											
13+##75#####											
14+##74#####											
15+##70#####											
16+##66#####											
17+##55#####											
18+##43#####											
19+##35#####											
20+##30#####											
21+##24#####											
22+##16#####											
23+#####	9										

Usage for : irca2a

Model 2 usage : (192 available) from 1 @ 02:11 to 146 @ 13:26

Model 4 usage : (4 available) from 0 @ 00:01 to 2 @ 10:01

Model 5 usage : (48 available) from 0 @ 00:36 to 4 @ 09:41

Usage for : irca3a

Model 2 usage : (192 available) from 1 @ 02:08 to 147 @ 13:32

Model 4 usage : (4 available) from 0 @ 00:02 to 3 @ 10:22

Model 5 usage : (48 available) from 0 @ 00:02 to 4 @ 11:02

Usage for : irca4a

Model 2 usage : (192 available) from 2 @ 02:08 to 147 @ 13:32

Model 4 usage : (4 available) from 0 @ 00:02 to 3 @ 08:47

Model 5 usage : (48 available) from 0 @ 00:02 to 4 @ 10:58

Usage for : irca5a

Model 2 usage : (192 available) from 1 @ 02:11 to 146 @ 13:12

Model 4 usage : (4 available) from 0 @ 00:01 to 3 @ 10:32

Model 5 usage : (48 available) from 0 @ 00:01 to 4 @ 11:22

Usage for : irca6a

Printing is also distributed on these systems. IBM systems are able to print to a variety of remote printers. Most of these printers are located someplace near the Sun workstations, connected to a TCP/IP-based Sun server. The IBM system is told to spool all print jobs through the gateways, where they end up on a Unix-based print server. There, the print daemon moves the files to a directly attached printer or to an auxiliary print server in another facility. Some printers are located at the other side of a facility from a print server. To handle these systems, many printers are attached to a terminal server. The terminal server then maintains a session with the print server over the network.

There are also large amounts of corporate data that are spooled onto database or file servers on a nightly basis. This data is used for applications such as a distribution control system or human resources applications. Moving the data into the workstation-based environment means that significantly more productive tools can be developed and used than within the basic 3270 terminal.

### *Town and Country*

Another approach to IBM connectivity is the Town and Country Building Society located in the United Kingdom. A building society is somewhat akin to the United States savings and loan industry, with the noticeable difference that building societies have managed to remain solvent.

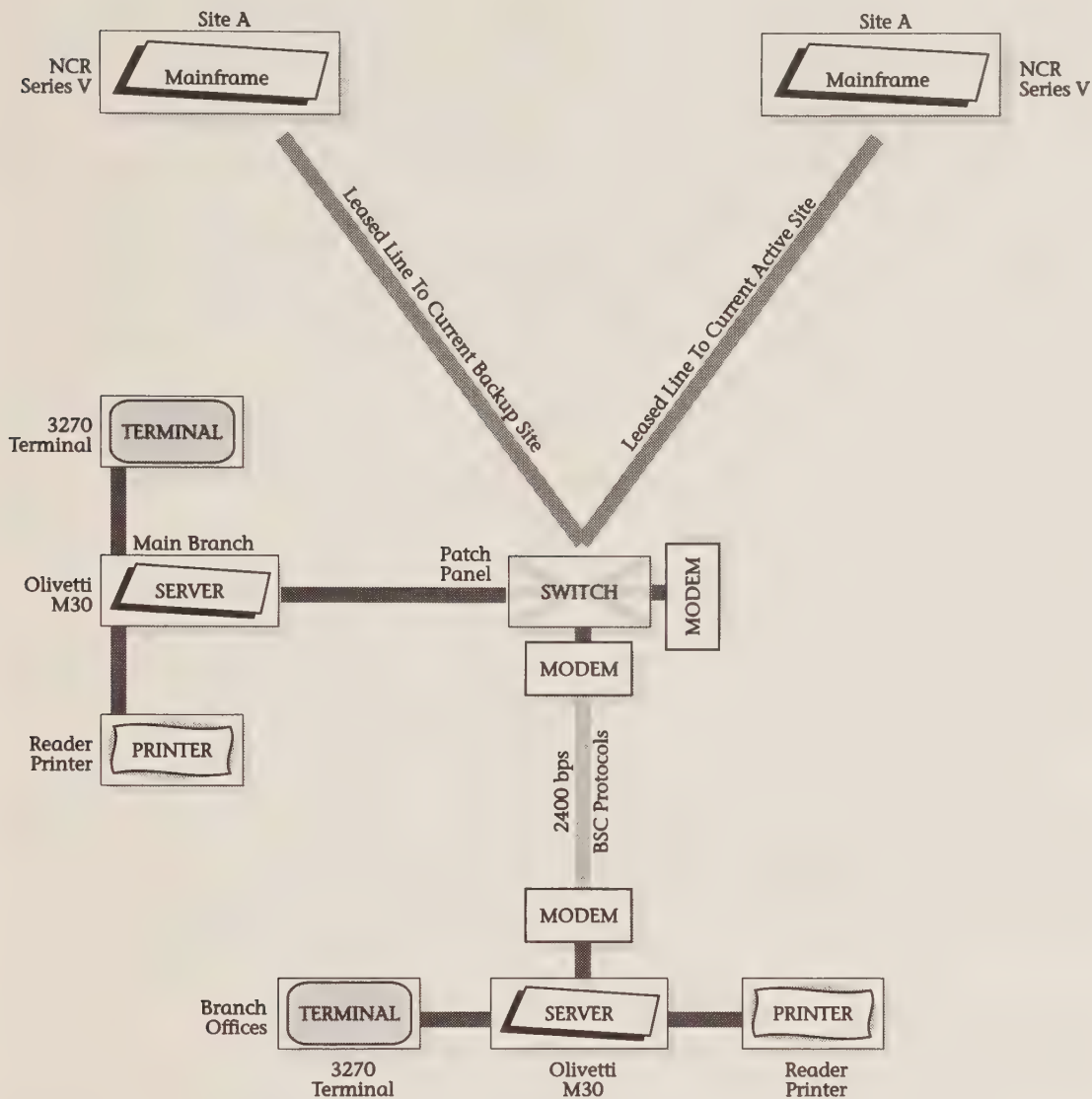
In the mid-80s, this organization was using very traditional applications based on 3270 terminals on an NCR Series V mainframe system. Dual main processing sites provide the bulk of the processing, with one site acting as the active site and the other as the hot backup. They switch roles between the main sites periodically to provide operating experience to both facilities.

Figure 16-6 shows the initial configuration for this environment. Control at each of the branch systems was provided by an Olivetti M30 processor, to which was attached a terminal and reader/printer devices. All the branch offices used 2400 bps modems and BSC/RJE protocols to the main branch.

At the main branch was a patch panel which linked all the traffic together and sent it to the appropriate processing site. Should one mainframe go down, traffic could be rerouted to the hot backup site.

The problem with this setup is that the M30 processor is not the world's most modern processor. The goal was to move toward some more standard equipment in the branches and possibly even provide more modern applications (such as word processors).

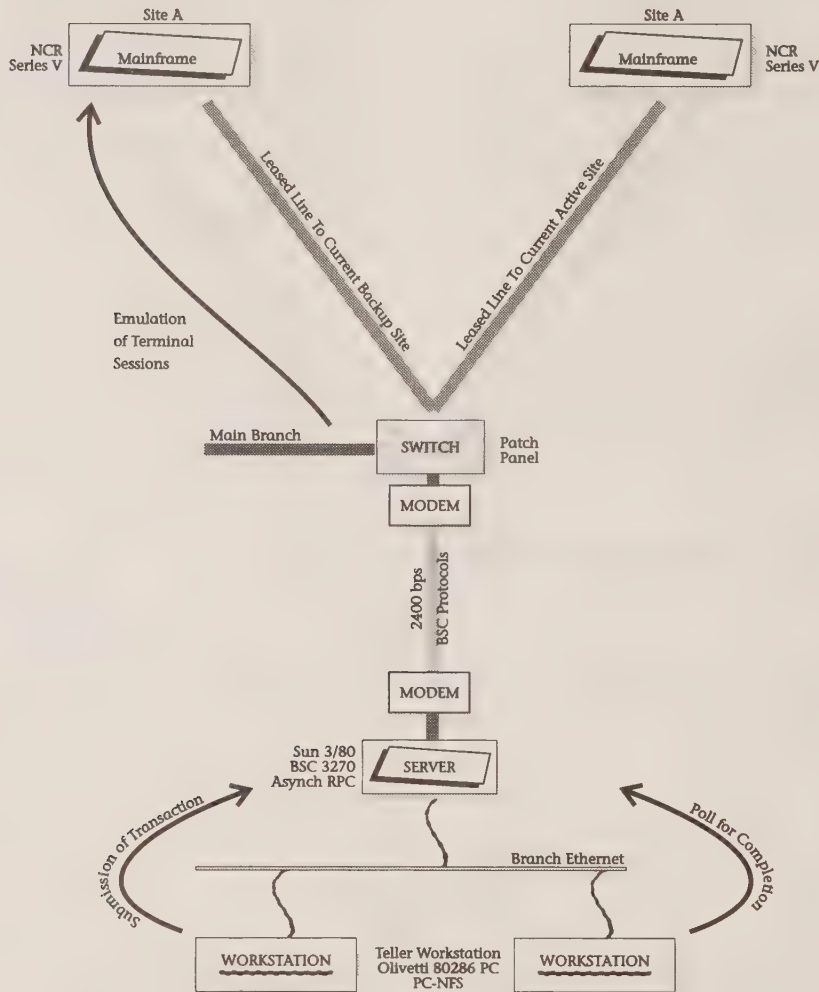
An equally important goal in such an environment is making sure the current applications keep on running. This is fairly crucial with a bank—imagine telling customers they can't have their money "because there are some bugs in our new computer software." Such a situation is considered suboptimal in much of the financial industry.



16-6 Town and Country, Phase 1

Figure 16-7 shows how a gradual transition was made to a more modern environment. An RPC-based program was written to run on a Sun server that emulated 3270 terminals. Then, a series of PC systems running PC-NFS were provided for tellers, loan officers, and other branch employees.

The PC system in its basic mode ran simple 3270 emulation software. That software would submit a transaction to the Sun server. The server would then send that transaction up the mainframe system using the same BSC/RJE protocols as before.



## 16-7 Town and Country, Phase 2

The server kept a status area for each of the terminals. Periodically, when a workstation had a transaction outstanding, it would poll for completion and then retrieve the results. Notice that the server is stateless: it is up to the PC to remember that it had a transaction outstanding, simplifying error recovery.

Notice also that this approach is modular. From the patch panel for the multidrop line on down, the system is transparent to the existing environment. Branches can be replaced one at a time and coexist with the existing system.



An immediate benefit for this stage was that the user terminal was now a PC, allowing applications like Lotus and word processing. Each teller system was an Olivetti 290 PC, Intel 80286-based PC with an internal 20-Mbyte hard disk and a 1.44-MByte 3.5 inch floppy. Each PC had a VGA color monitor, 1 MByte of RAM in the base, 4 Mbytes of expanded memory, serial and parallel ports, and was considered a small footprint system (by 1988 PC standards).

What is interesting is that these PC systems had previously been procured for another approach to the problem, one that had not succeeded. In that system, dbase-based applications would have used a proprietary gateway and some PC-based networking scheme.

The next step was to move away from the Bisynch protocols to a more modern approach to data communications, shown in Figure 16-8. The program that emulated terminals was moved up to the main processing sites and moved onto a Sun-3/470. This server system had two 100-kbps links into an X.25 network as well as 12 ports to the NCR mainframe, each capable of handling 32 simultaneous sessions.

The machine in the branch offices stayed the same but no longer ran the RPC program. Instead, this branch server became a simple Internetwork Router with SunNet X.25 software. The workstation software didn't need to be changed; it simply used the address of a different server.

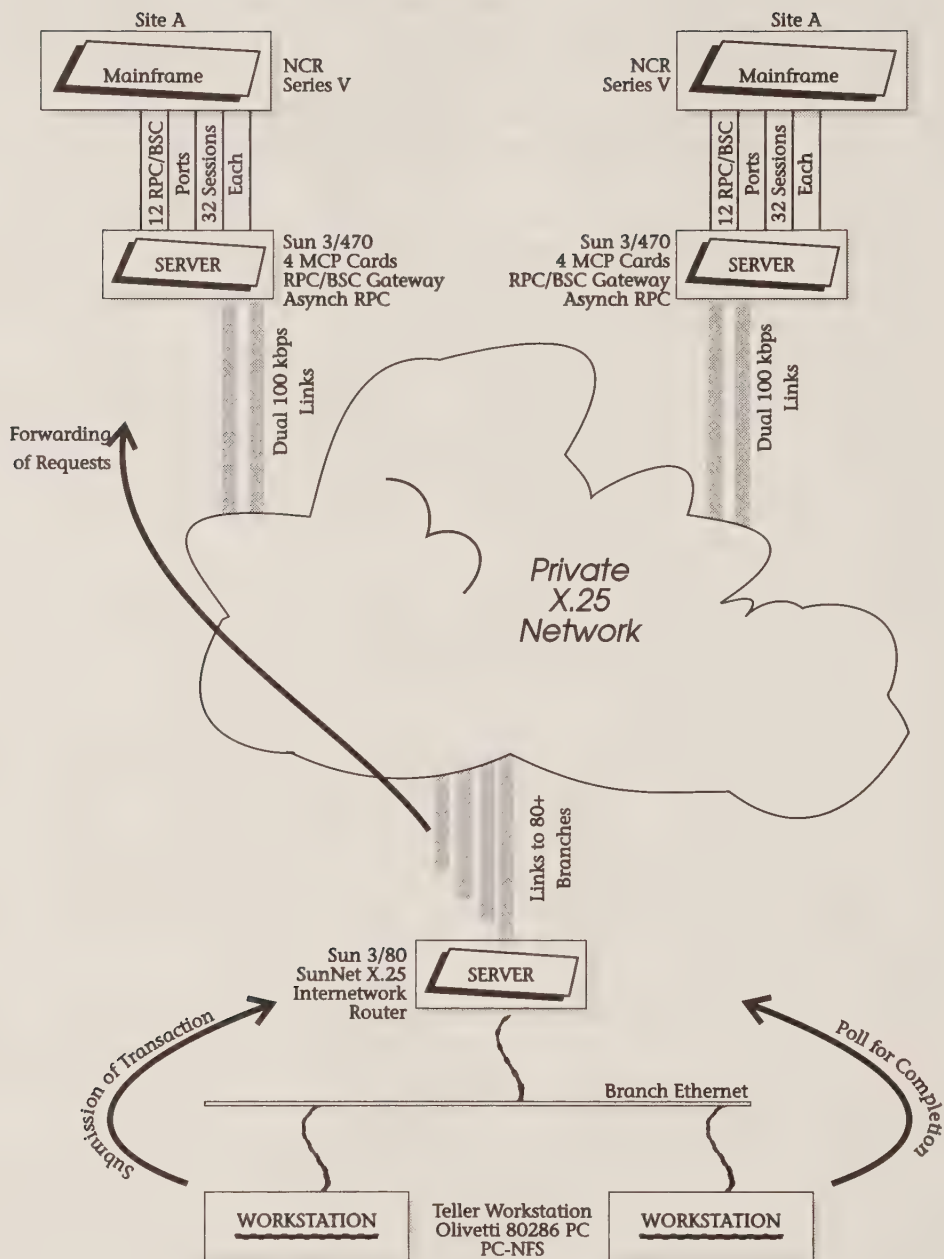
At this point, the mainframe systems are running unchanged, but the distribution to the branches is significantly faster and more flexible. The next step, not shown on the diagrams, is to supplement the services of the NCR Series V with a more modern approach. This involves adding a Sun-3/490 running the Sybase relational database software. The Sun-3/470 still acts as the gateway at data processing, forwarding requests to the Sybase server or the NCR mainframe. The Sybase system can be used for applications like loan processing.

Note that the key here was that the existing network was allowed to continue and migration was gradual. The branches were able to slowly switch over. After that was completed, the data processing gateways were moved into place, and finally, Unix-based servers added.

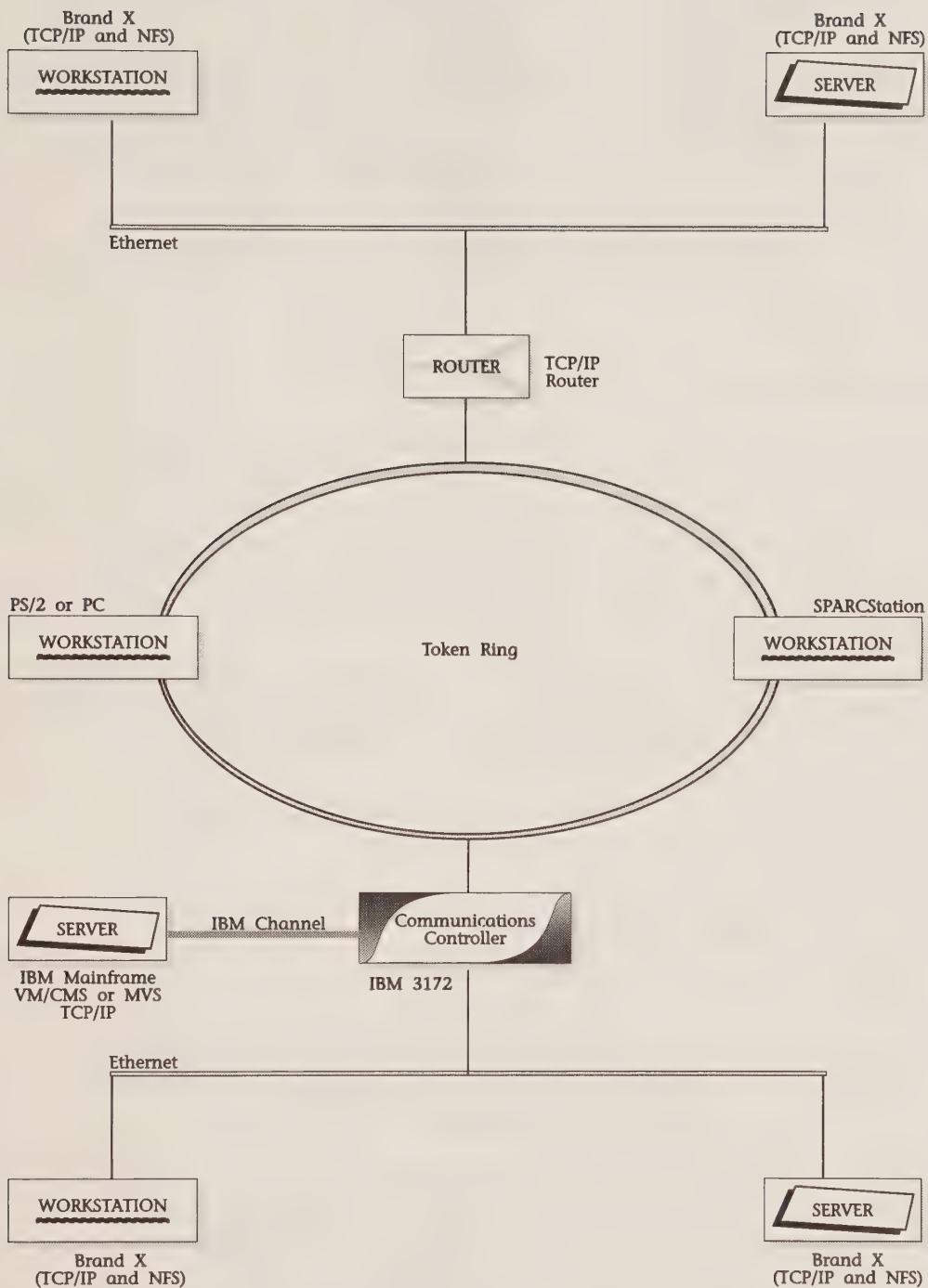
Had this system been set up from scratch (and large financial institutions are rarely in that position), the entire system could have been based on the TCP/IP protocols along with NFS implementations for either the VM or MVS operating systems.

Figure 16-9 shows this last connectivity option. Notice that the mainframe system is connected to an IBM 3172 cluster controller, which in turn has connections to token ring and Ethernet. At this point, the entire environment is homogenous and the IBM is simply another server.

Sun's IBM connectivity played an integral part in the development of MVS/NFS. All engineers on the project had a Sun workstation on their



16-8 Town and Country, Phase 3



## 16-9 Integrating IBM Using TCP/IP

desks. Interaction with the mainframe was through 3270 terminal emulation or the Network Job Entry (NJE) software. The 3270 terminals were often used to grab the MVS console to perform systems work. The NJE-based Remote Job Entry was used to upload source code to the mainframe and build the MVS/NFS application under the control of Unix makefiles.

Even source code control was kept off the mainframe, using the Unix SCCS utility instead. This includes the 6000 lines of 370 assembly language. The development team even went so far as to emulate MVS routines on Unix so that higher-level MVS functions could be debugged on the workstation, leaving only assembly language routines that needed to be debugged on the mainframe.

### The Rest of the World

Now that we have the mainframe systems out of the way, let us look at integrating a few other pieces into the network. DEC and Sun equipment, for example, are often found together in the same networking environment.

DEC equipment has a native networking environment based on DECnet. DECnet Phase IV is a proprietary environment but has many similarities to TCP/IP. DECnet has the Data Access Protocol (DAP), for example, a protocol that overlaps with the File Transfer Protocol.

Integration of the two environments is performed at several levels. Both DECnet and Sun equipment can share the same data links, including FDDI, Ethernet, Ultranet networks, or X.25. It is possible to run parallel networks and simply share the underlying media.

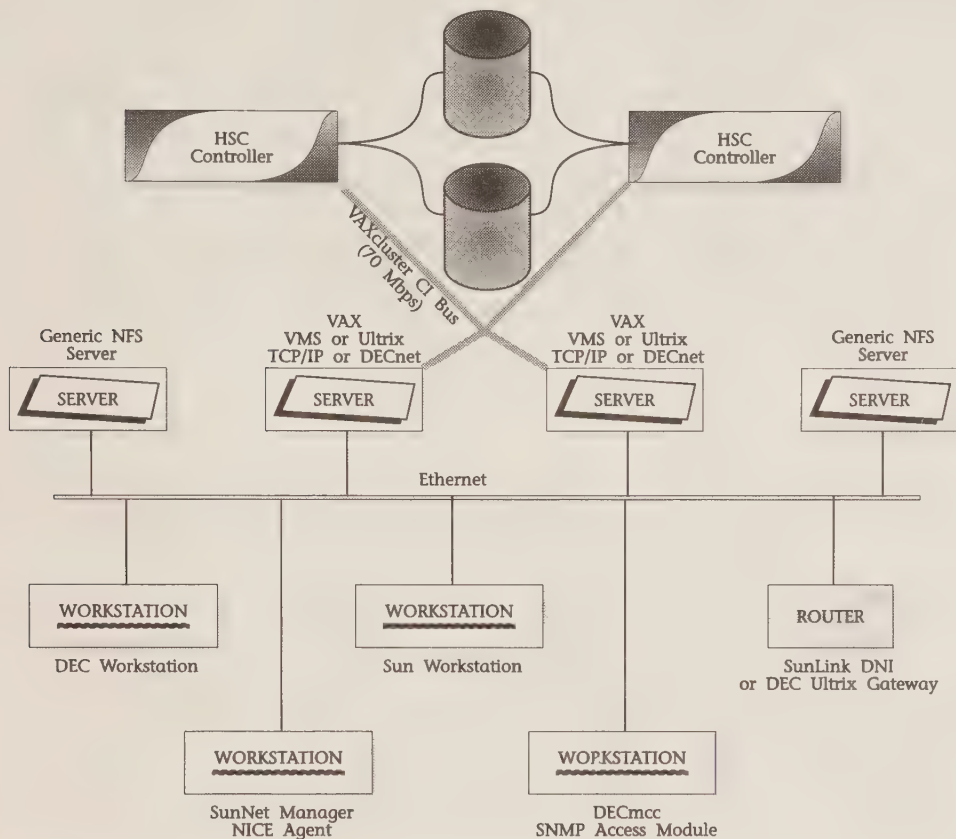
Figure 16-10 shows a small network configuration with four kinds of equipment:

- Workstations
- Network management consoles
- Servers
- Gateway systems

If we are running parallel universes, the gateway system isn't being used. The DEC workstation would use DECnet and talk to the VAX systems. The DECnet nodes would be able to communicate only with each other and run DECnet applications such as DAP.

The other universe, the TCP/IP world, would be the Sun workstation and whatever NFS and TCP/IP servers are on the network. Again, the members of this universe would be able to talk only to each other and would run only TCP/IP applications like NFS and FTP. Notice that the two universes share the services of the underlying Ethernet. Ethernet is a general-purpose service provider, simply delivering packets to their destination.





16-10 Interconnection with DECnet

To link the two universes together, we can take two approaches. A simple one is to switch the whole DECnet world over to TCP/IP. DEC supports TCP/IP as a networking option for both VMS and Ultrix operating systems.

DEC also supports the NFS protocols on their VAX Cluster environment. VAX Cluster systems can connect several different large minicomputers together over a 70-Mbps bus to dedicated disk controllers. The disk controllers are in turn connected to dual-ported disk arrays, providing large amounts of redundant storage.

A cluster makes a good NFS server for very large amounts of data. Another approach to this problem that we saw in Chapter 6 is to use dedicated NFS servers from companies like Auspex and Epoch (or dedicated Sun servers with fast disk drives). A second approach to integration is to have the DEC systems speak two network languages: TCP/IP and DECnet. Using this approach, the server is able to provide its files, programs, printers, and other resources to both environments. Finally, we have the third approach

of using a gateway. A gateway simply links the two environments together at a certain point, usually by bridging two applications.

Both Sun and DEC sell software to interconnect the two environments (as do other vendors). With Sun, the software is SunLink DNI, which acts as a DECnet Phase IV end node and provides TCP/IP users access to protocols such as the DECnet NICE (network management) and DAP protocols. The equivalent DEC software is the Ultrix gateway, which links DECnet mail, file transfer, and terminal emulation software to their TCP/IP equivalents. One of the more important applications for many is network management. Both DECMCC and SunNet Manager support management of the other environments.

### *DECnet Phase V*

Phase V of DECnet is an extension of the proprietary DECnet Phase IV protocols with the addition of protocols from the OSI and TCP/IP protocol suites. Phase V is really three different networks spliced together.

The way the networks are spliced together provides several entry points. For file access, for example, the DECnet node could be running FTAM, FTP, or DAP (or NFS or a database package).

A simple way to cut through the maze when connecting to this environment is to use OSI as a common ground. Phase V supports basic OSI protocols such as FTAM and X.400. SunLink OSI and SunLink Message Handling System (MHS) software support these two protocols, so they can be used as a common ground.

Another approach to connecting the environments together is to use TCP/IP and NFS as the common ground. This works as long as the user has not been tied into DEC proprietary protocols such as LAT, the naming service, or DAP.

### *OSI*

OSI connectivity for Sun equipment is basically provided by picking a key subset of OSI protocols (see Fig. 16-11). The SunLink MHS software is the CCITT X.400 message handling system gateway, which links the native SMTP protocols to the X.400 world.

FTAM and other services are provided as part of the basic SunLink OSI software systems. This software, in turn, runs over X.25, Ethernet, token ring, or FDDI data links. To connect to another vendor's OSI implementation, find a common denominator among the data links.

Another approach to OSI connectivity is using an interim solution which comes from the ISO Development Environment (ISODE). Figure 16-12 shows how a TCP/IP-based Internet can be used to transport data for OSI applications.

CCITT X.400	Custom Programs	ISO 8571 File Transfer, Access, Management (FTAM)		
		ISO 8650 Association Control Service Element (ACSE)		
		ISO 8823 Presentation Protocol		
ISO 8327 Session Protocol				
ISO 8073 Transport Classes 0 and 2	ISO 8073 Transport Class 4			
ISO 8878 Connection- Oriented Network Service (CONS)	ISO 8473 Connectionless Network Protocol (CLNP)			
ISO 8208 X.25 Packet Level Protocol (PLP)				
ISO 7776 Link Access Protocol B (LAPB)	ISO 8202/2 Logical Link Control (LLC) Classes 1 and 2			
X.21 bis	ISO 8802/3 10-Mbps CSMA/CD	ISO 9314 FDDI	ISO 8802/5 4-Mbps Token Ring	

## 16-11 Sun OSI Support

As we saw in Chapter 4, RFC 1006 defines how TCP provides the services we would expect to find in an environment based on the connection-oriented network service. If we add a small amount of header information, the TCP/IP environment is considered to be just another hop in the path.

In order to run OSI applications in a TCP/IP environment, software like ISODE (or one of the commercial derivatives of this approach) is used. Applications such as X.500 services are run on top of the OSI/TCP hybrid stack.

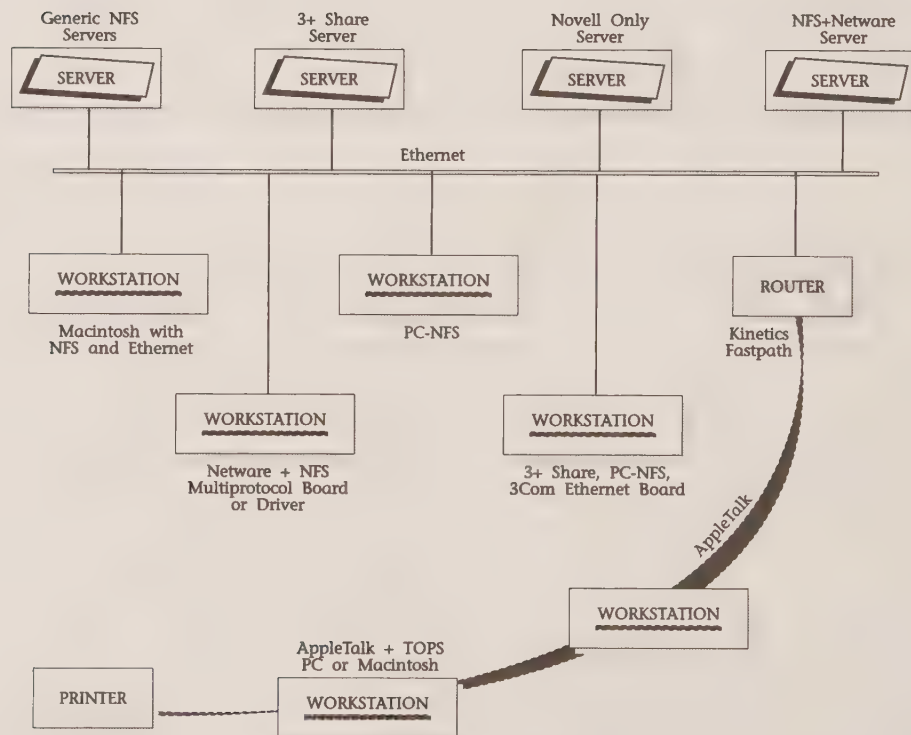
### The PC

We're still missing one piece, the PC. Again, a simple solution is to run a single protocol, such as PC-NFS or any of the other TCP/IP-based solutions available for the DOS and Macintosh operating systems.

For the same reason that we couldn't make all IBM mainframes stop speaking SNA, we are unfortunately unable to make all PC systems stop

Local File System	Local File System		Local File System
ISO FTAM	ISO FTAM		FTAM
ISO ACSE	ISO ACSE		ACSE
ISO Presentation	ISO Presentation		ISO Presentation
ISO Session	ISO Session		ISO Session
ISO Transport	ISP Transport	RFC 1006	RFC 1006
		TCP	TCP
CLNP		IP (with or without X.25)	
LAN or WAN Data Link		LAN or WAN Data Link	
LAN or WAN Media		LAN or WAN Media	

## 16-12 TCP/OSI Bridging



## 16-13 Interoperability with PC Systems



using protocols such as NetWare. For the Macintosh workstations, there are the same choices: AppleTalk or TCP/IP.

As before, we can run parallel universes. A Novell NetWare server, for example, can be used to provide Novell data to Novell workstations while a parallel TCP/IP universe revolves around it (see Fig. 16-13). It is possible to have a PC run both universes at once using a multiprotocol driver or (preferably) a multiprotocol board.

Another approach to integrating the NetWare environment is to make the server support both protocols. The server runs a NetWare Loadable Module as part of the Novell Operating System. This turns the Novell server into a generic NFS server.

With the Macintosh, there are more compelling reasons to run both TCP/IP and AppleTalk. AppleTalk is built into many devices including all Macintosh systems and many laser printers, meaning that it is a much cheaper solution than Ethernet.

Gateway systems exist between the TCP/IP world and an AppleTalk environment. Kinetics, for example, uses a Fastpath gateway system to link the two environments. Tops has software that allows support for both AppleTalk and TCP/IP environments.

Again, a simpler solution is to put the workstation right on the Ethernet and have it run TCP/IP. A very high level of transparency can be had using services like X Windows and NFS. Any program on any server can be run by the workstation without having to worry about artificial walls put up by the parallel universes approach: "Sorry you can't have that data with this program on that machine."

### For Further Reading

Malamud, *Analyzing DECnet/OSI Phase V*, Van Nostrand Reinhold (New York: 1991).

———, *STACKS: Interoperability in Today's Computer Networks*, Prentice Hall (Englewood Cliffs, NJ: 1991).

Sun Microsystems, *SunLink BSC RJE User's Guide*, Part No: 800-3033-10, Revision A of 18 October 1988.

———, *SunLink CG3270 User's Guide*, Part No: 800-4962-10, Revision A of 26 October 1990.

———, *SunLink NJE User's Guide*, Part No: 800-3009-05, Revision 50, 19 August 1988.

———, *SunLink DNI Programmer's Guide*, Part No: 800-5375-10, Version A, 15 March 1991. SunLink DECnet Interconnection.

———, *SunLink DNI System Administrator's Guide*, Part No: 800-5374-10, Version A, 15 March 1991.

———, *SunLink DNI User's Guide*, Part No: 800-5373-10, Version A, 15 March 1991.

———, *SunLink OSI System Administration Guide*, Part No: 800-3214-10, Revision A of 21 July 1989.

———, *SunLink OSI Programmer's Manual*, Part No: 800-3215-10, Revision A of 21 July 1989.

———, *SunLink SNA Peer-to-Peer API Programmer's Guide*, Part No: 800-3045-10, Version A of 7 October 1988.

———, *SunLink SNA Peer-to-Peer DIA Programmer's Guide*, Part No: 800-3044-10, Version A of 7 October 1988.

———, *SunLink SNA Peer-to-Peer System Administration Guide*, Part No: 800-3043-10, Version A of 7 October 1988.



# Glossary





# Glossary

1003.4	POSIX real-time extensions.
1003.6	POSIX security extensions.
1003.7	POSIX system management extensions.
1003.8	POSIX networking extensions.
10BASE2	10 Mbps/baseband/200 meters. IEEE standard for ThinWire Ethernet.
10BASE5	10 Mbps/baseband/500 meters. IEEE standard for Thick-Wire coaxial Ethernet.
10BASET	10 Mbps/baseband/twisted pair. IEEE standard for twisted pair Ethernet.
2780/3780	A model of remote batch terminals used in IBM bisynchronous environments. Bisynch gateways are often referred to as 2780/3780 emulators.
3+	3Com networking products.
3090	IBM top of the line mainframe, sometimes called Sierra.
3270	A series of terminals used in IBM environments.
3270 Display Stations	Terminals for IBM mainframe computers.
3274	A cluster controller for IBM equipment, often called a terminal concentrator.

3480	IBM tape cartridges.
370 architecture	IBM architecture for mainframe computers, including the 3090 processors.
3705	An IBM communications controller. A Physical Unit (PU) type 4 in SNA, used to connect token ring networks, cluster controllers, non-IBM SNA gateways, and other devices to a PU Type 5 host.
3725	<i>See 3705.</i>
3Com	A communications company known for Ethernet controllers and PC-based networking equipment. Merged with Bridge Communications.
4010/4014	A series of Tektronix graphics terminals. A de facto standard for graphics terminals.
4.3BSD	<i>4.3 Berkeley Software Distribution.</i> The current version of the Berkeley family of Unix products.
4GL	<i>See fourth-generation language.</i>
5250	IBM terminal used on the System/3X line.
802.2	IEEE standard for the Logical Link Control.
802.3	IEEE standard for CSMA/CD (Ethernet) medium access method.
802.4	IEEE standard for the token bus medium access method.
80x86	Family of microprocessors made by Intel used in the PC line. The PC/AT uses the 80286 chip. More modern systems, including the PS/2, use the 32-bit 80386 and 80486 chips.
9370	Mid-range IBM processor.


---

ABM	<i>See asynchronous balanced mode.</i>
abstract data type	Concept used in programming and database systems. An abstract data type is a user-defined data type that hides the internal details of its manipulation. For example, date, an abstract data type, is stored internally as an integer but is manipulated externally through date-specific representations and operations.
AC	<i>Access control.</i> Token ring field that holds the priority and reservation bits for a token or data packet.
Access Control Set	A set of identifiers of objects and their associated rights to some entity (e.g., a node or a file).
access method	A means of accessing information in a file. Btree is an example of an access method.
ACK	<i>Acknowledge.</i> A network packet acknowledging the receipt of data.
ACL	<i>Access Control List.</i> A security feature in operating systems that allows security on objects to be specified as a list of permitted actions for particular lists of users.
ACM	<i>Association for Computing Machinery.</i>
ACSE	<i>See Association Control Service Element.</i>
ACSNET	<i>Australian Computer Science Network.</i> ACSNET is the principal electronic mail system for the computer science and academic research community in Australia, connecting about 300 sites. It works similarly to UUCP. [RFC 1168]
Action Technologies	Makers of the Message Handling Service bundled into Novell as NetWare MHS.
active converter	A device used to convert from one communication signaling interface to another.
active device	A device with its own power source.
Active Monitor Present	Packet issued every 3 seconds by the active monitor on a token ring.

activity daemon	Process that keeps a list of active agent requests that have been sent from the host where the daemon is running. The daemon periodically queries the agents to make sure they are still servicing their request. [SunNet Manager]
AD	<i>Administrative domain.</i> A group of hosts, routers, and networks operated and managed by a single organization. Routing within an administrative domain is based on a consistent technical plan. An administrative domain is viewed from the outside, for purposes of routing, as a cohesive entity, of which the internal structure is unimportant. Information passed by other administrative domains is trusted less than information from one's own administrative domain. [RFC 1136]
ADDMD	<i>Administration Directory Management Domain.</i> An X.500 directory management domain maintained by a public telecommunications entity which is a member of the CCITT.
address	There are two separate uses of this term in internet networking: "electronic mail address" and "internet address." An electronic mail address is the string of characters that you must give an electronic mail program to direct a message to a particular person. [RFC 1177] <i>See also internet address.</i>
Address Resolution Protocol	A TCP/IP protocol to translate an IP address into a MAC address (e.g., an Ethernet or other subnetwork address).
address space	A collection of addresses that form a unified collection such as an internetwork.
addressing authority	The group responsible for assigning addresses within a domain.
ad hoc	Latin phrase meaning for a specific instance. Used in computing to refer to functions not previously planned.
ADMD	<i>Administration Management Domain.</i> X.400 message-handling system concept. Countries will typically be ADMDs. They might then cede some management authority to a private management domain (PRMD).



<b>administrative domain</b>	A collection of End Systems, Intermediate Systems, and sub-networks operated by a single organization or administrative authority. The components which make up the domain are assumed to interoperate with a significant degree of mutual trust among themselves, but interoperate with other administrative domains in a mutually suspicious manner. [RFC 1136]
<b>Adobe Systems</b>	The company that defined the Postscript Page Description Language.
<b>ADT</b>	<i>See abstract data type.</i>
<b>Advanced Program-to-Program Communication</b>	Protocol used for peer-to-peer communication in IBM's System Network Architecture.
<b>advertising</b>	The process by which a service makes its presence known on the network. Typically provided through some form of LAN-based multicast.
<b>AES</b>	<i>See Application Environment Specification.</i>
<b>AFI</b>	<i>Authority and Format Identifier.</i> Part of an OSI address which signals what type of address follows the AFI.
<b>AFP</b>	<i>See AppleTalk Filing Protocol.</i>
<b>aged packet</b>	A routing layer packet that has exceeded the maximum number of visits.
<b>Agenda</b>	Lotus software used for time management.
<b>agent</b>	Network management term for the portion of an entity that responds to management functions.
<b>agent schema</b>	A formatted description of the groups and attributes available from an agent, with an enumerated list of possible agent errors. This description enables manager processes to make requests for specific groups and attributes. [SunNet Manager]
<b>Agent Services</b>	The function library that allows agents and managers to communicate, as viewed by the agent. [SunNet Manager]

<b>aggregate</b>	A function in a query language used to perform an operation on several rows of data. Sum is an example of an aggregate.
<b>AIX</b>	<i>Advanced Interactive Executive.</i> IBM's version of Unix.
<b>AK TPDU</b>	<i>Acknowledge Transport Protocol Data Unit.</i> A protocol data unit type in the OSI transport service.
<b>alias</b>	A name that is translated into another name. A DNS soft link is an example of an alias.
<b>allocation</b>	A concept used in the transport layer protocols. An allocation is the amount of unacknowledged traffic that may be outstanding at one time.
<b>ALTERNET</b>	A commercial TCP/IP-based network run by the people who run the UUnet commercial UUCP service.
<b>American National Standards Institute</b>	A private organization that coordinates some United States standards-making. Represents the United States to the International Standards Organization.
<b>American Wire Gauge</b>	Standard used to describe the size of a wire.
<b>A-mode</b>	<i>Asynchronous Mode.</i> Used in the OSI Virtual Terminal protocols. A model for terminal operation where either side may communicate at any time.
<b>AMP</b>	<i>See Active Monitor Present.</i>
<b>annular mark</b>	The mark on an Ethernet coaxial cable that identifies the 2.5-meter separation required for tap points on the cable.
<b>ANSI</b>	<i>See American National Standards Institute.</i>
<b>APDU</b>	<i>Application Protocol Data Unit.</i> An application layer PDU.
<b>API</b>	<i>Application programming interface.</i> The function library interface an application can call to perform some service. [SunNet Manager]
<b>APPC</b>	<i>See Advanced Program-to-Program Communication.</i>
<b>Apple</b>	Maker of AppleTalk and the Macintosh.
<b>aggregate</b>  <b>Apple</b>	

<b>AppleShare</b>	Program that allows different kinds of computers to provide or take advantage of AppleTalk resources such as printing.
<b>AppleTalk</b>	Apple's network protocol.
<b>AppleTalk Filing Protocol</b>	The protocol in AppleTalk used for remote access to data.
<b>AppleTalk Session Protocol</b>	AppleTalk session layer protocol.
<b>AppleTalk Transaction Protocol</b>	AppleTalk transport layer protocol.
<b>application</b>	A program that performs functions for a user. Order entry system or word processors are both examples of applications.
<b>Application Control Services</b>	NAS concept. The rules used for application-to-application interaction in a NAS environment.
<b>Application Environment Specification</b>	A goal of the Open Software Foundation.
<b>application layer</b>	The top layer of the network protocol stack. The application layer is concerned with the semantics of work. For example, getting a certain record from a file by key value on a foreign node is an application layer concern. How to represent that data and how to reach the foreign node are issues for lower layers of the network.
<b>application process</b>	A component of a distributed application executing on a single computer. Synonymous with process. [Netwise RPC Tool]
<b>Application Programming Interface</b>	Specification of the calling structure between two programs, usually between a general application program and a specific support service, such as communications support.
<b>application terminal</b>	An LAT object accessed by an application on a slave node. For example, a printer attached to a terminal server would be the application terminal for the print spooler.

ARCnet	Hardware and software data link components manufactured by Datapoint and other companies that allow computers to form a 2.5-Mbps local-area network with a star topology.
Areas	A term used in the routing layer. Level 1 routers are used to route within a single area. Level 2 routers route between areas. Up to 1023 nodes may be in an area, up to 63 areas in a DECnet. OSPF uses a similar concept.
argument PDU	A PDU containing the user-data portion of a request PDU. [Netwise RPC Tool]
ARP	<i>See Address resolution protocol.</i>
ARPA	<i>Advanced Research Projects Agency.</i> The former name of what is now called DARPA.
ARPANET	<i>Advanced Research Projects Agency Network.</i> A Department of Defense sponsored network of military and research organizations. Replaced by the Defense Data Network.
ARPOSE	<i>Specification de l'Architecture et des Protocoles OSI permettant de rendre le service le services Session.</i> OSI Profile for France.
array	A data structure used in programming.
AS	<i>Autonomous System.</i> A set of routers under a common technical administration.
AS/400	<i>Application System/400.</i> IBM mid-range computer replacing the System/38 and System/36 product lines.
ASCII	<i>American Standard Code for Information Interchange.</i> A standard character set that assigns an octal sequence to each letter, number, and selected control characters. The other major encoding standard is EBCDIC.
ASE	<i>Application Service Element.</i>
Ashton-Tate	Makers of dBase and Framework data management products.
ASN.1	<i>Abstract Syntax Notation One.</i> OSI presentation layer protocol.



ASP	<i>See AppleTalk Session Protocol.</i>
assigned numbers	Those numbers officially assigned as part of the Internet standards.
Association Control Service Element	Core set of facilities in the OSI application layer which allow application entities to form an association.
AST	<i>Asynchronous system trap.</i> A concept used in the VMS lock manager. An AST is a request to be notified when a certain event occurs. In the case of the lock manager, an AST can be set so that a process holding a lock on a resource is notified if another process tries to take an incompatible lock on the same resource.
Async	<i>Asynchronous.</i> A data transmission method that sends one character at a time. Contrasted with the synchronous methods, which send a packet of data and then resynchronize their clocks. Asynchronous also refers to commands, such as in a windowing environment, that may be sent without waiting for a response from the previous command.
asynchronous	FDDI term for data transmission where all requests for service contend for a pool of ring bandwidth.
asynchronous balanced mode	A term used in the IEEE 802.2 standard to refer to a situation in which only one side of the connection can send.
asynchronous communication	Communication in which every byte is sent individually. Synchronous communication (e.g., X.25 or Ethernet) bunches several pieces of data into a frame or packet.
asynchronous event	Events occur asynchronously on a system when the user cannot predict which one will happen next.
asynchronous processing	A method of operation wherein the calling procedure's execution is not suspended after calling another procedure and while waiting for a response. <i>See synchronous processing.</i> [Netwise RPC Tool]
AT	<i>Advanced technology.</i> Shorthand for the IBM PC/AT or any other Intel 80286-based processor that runs the DOS operating system.
ATM	<i>Asynchronous Transfer Mode or Automated Teller Machine.</i> Asynchronous transfer mode is also known as "fast packet." Method for dynamic allocation of bandwidth on a cell basis.

ATP	<i>See AppleTalk Transaction Protocol.</i>
AT&T	<i>American Telephone and Telegraph.</i> Provider of long-distance service and computing company known for the Unix operating system.
attenuation	The level of signal loss, usually expressed in units of decibels.
attenuation characteristic	As a signal propagates on a cable, it gets weaker, or attenuates. The attenuation characteristic of the medium is the rate at which it gets weaker.
attribute	A “perceived property” of some entity that can be read and may be modified. Attributes are used in network management as well as in the naming service. An example would be the password attribute of the object user. In a relational database, attribute is another name for a column in a table. In a data dictionary or other information model, an attribute is attached to a relationship or entity.
attribute group	A named collection of attributes.
attribute name	The name of an attribute as defined by the agent schema. [SunNet Manager]
AUTH	<i>Authentication Service.</i> Service defined in RFC 931
authentication	The function of verifying the identity of a person or process.
authorization	Determining if a person or process is able to perform a particular action. Contrast with authentication.
autobaud	The ability of a modem on the receiving end of a call to automatically detect the speed of transmission used by the calling modem.
AWG	<i>See American Wire Gauge.</i>
b	<i>Bit—binary digit.</i> The smallest amount of information which may be stored in a computer. [RFC 1177]
B	<i>Byte.</i> One character of information, usually 8-bits wide. [RFC 1177]

backbone	A networking term used to refer to a piece of cable used to connect different floors or departments together. Contrasted with a departmental network or work area network.
backup	Making a copy of stored information to use in case the original repository (usually a disk drive) becomes corrupted. To be contrasted with the alternate meaning of the word, "to overflow," which is usually used in the context of plumbing and sewage.
balun	<i>Balanced/unbalanced.</i> An adapter between two different pieces of physical media that adjusts for the difference in impedance.
bandwidth	The amount of data that can be moved through a particular communications link. Ethernet has a bandwidth of 10 Mbps.
Banyan	Maker of VINES, a PC network.
baseband	Coaxial cable implementation of Ethernet. Also known as ThickWire.
BASIC	<i>Beginner's All-purpose Symbolic Instruction Code.</i> A programming language.
BAT	<i>See binary absolute time.</i>
batched calling style	A special case of the no-reply calling style in which the transmission or the delivery of the request is not required to take place immediately but may be combined with subsequent requests. [Netwise RPC Tool]
baud	A term used with older (slow) modems to refer to each modulation of an analog signal. A 300-baud signal modulates 300 times per second. A more accurate term for faster modems is bits per second, as several bits can now be carried on one modulation of the signal.
bayonet nut connector	Connector type used for 10BASE5 coaxial cable. The term <i>bayonet</i> refers to the way the connector slides in and then twists to lock the connection.

BBN	<i>Bolt, Beranek, and Newman, Inc.</i> The Cambridge, Mass., company responsible for development, operation, and monitoring of the ARPANET, and, later, the Internet core gateway system, the CSNET Coordination and Information Center (CIC), and NSFnet Network Service Center (NNSC). [RFC 1177]
BBOARD	<i>Bulletin board.</i>
beacon	A token ring packet that signals a serious failure on the ring.
Beepy	Code name for Brian Pawlowski, a liveware project at Sun Microsystems.
BER	<i>Basic Encoding Rules.</i>
best-effort delivery service	A network module, such as the network layer IPX, will attempt to deliver data but will not try to recover if there is an error such as a line failure.
big endian	A computer that stores a multi-octet data structure with the lowest addressed octet as being the most significant. <i>See also endian and little endian.</i>
binary absolute time	Time specified as coordinated universal time (UTC), a time differential factor (TDF) and an inaccuracy term.
binary tree	Often referred to as a Btree. A storage structure with a dynamic index used for environments with frequent updates to data.
bind	An SNA term used to establish a session between two logical units.
bindery	A file on the NetWare operating system used to store management information such as user passwords.
binding	Concept used in remote procedure calls. Two remote programs bind with each other by starting a connection and then exchanging command requests. <i>See also RPC binding.</i>
binding class	A qualifier used for parameters or variables appearing in process-binding declarations. The binding classes are bind in, bind out, and bind in/out. [Netwise RPC Tool]



binding declaration	A statement in an RPC specification that provides binding information. This statement contains a parameter or external variable, its binding type, and binding class. [Netwise RPC Tool]
binding descriptor	A value of the Netwise-defined type <code>binding_desc</code> which contains information concerning a specific binding endpoint. [Netwise RPC Tool]
binding endpoint	The local representation of one end of a binding. [Netwise RPC Tool]
binding information	A data structure which can be used to identify a server application process on a network. [Netwise RPC Tool]
binding parameter	A parameter or external variable that is specified in a binding declaration. [Netwise RPC Tool]
binding styles	The four methods of implementing a binding supported by the Netwise RPC Compiler: non-persistent, persistent-open, persistent-use, and persistent-close. [Netwise RPC Tool]
binding type	One of the Netwise-defined C types used for parameters or external variables in binding declarations. The binding types are <code>binding_info</code> , used for new bindings, and <code>binding_desc</code> , used for existing bindings. [Netwise RPC Tool]
BIOS	<i>Basic input/output system.</i> The MS-DOS library of calls for access to data. Shields the application from the different types of physical disks.
Bisynch	A synchronous protocol used in older IBM teleprocessing environments. <i>See also</i> BSC.
bit mapped	A graphics term in which all bits of a display station are controllable; in contrast to a character-oriented terminal.
BITNET	<i>Because It's Time Network.</i> BITNET has about 2500 host computers, primarily at universities, in many countries. It is managed by EDUCOM, which provides administrative support and information services. There are three main constituents of the network: BITNET in the United States and Mexico, NETNORTH in Canada, and EARN in Europe. There are also AsiaNet, in Japan, and connections in South America. <i>See</i> CREN. [RFC 1177]
BLICN	A type of physical network.

block	A unit of I/O on computers. A block often ranges from 512 bytes to 8 kbytes.
blocking	The suspension of the execution of an application process until some specified condition is satisfied. Blocking occurs in synchronous processing. This term is frequently used to describe the suspension of execution of a client application process until a remote procedure returns. [Netwise RPC Tool]
blocking call	A procedure call in which the caller halts its own execution until the called procedure finishes.
BNC	<i>See bayonet nut connector.</i>
BNF	<i>Backus-Naur Form.</i> A way of specifying the syntax of a language.
BOOTP	<i>Bootstrap Protocol.</i> Protocol described in RFC 951 used for booting diskless nodes.
BOS	<i>Binary object sequence.</i> The lowest level of access to a PostScript Interpreter. Higher levels are PostScript programming library calls and ASCII interfaces.
bps	<i>Bits per second.</i> Transmission speed on modems, phone lines, and other data communications devices.
BRA	<i>See broadcast router adjacency.</i>
bridge	A device used to connect two separate Ethernet networks into one extended Ethernet. Bridges only forward packets between networks that are destined for the other network. Term used by Novell to denote a computer that accepts packets at the network layer and forwards them to another network.
broadband	An analog media similar to cable TV. Large bandwidth and very long distances make this media appropriate for campus settings. Used with various data link protocols including Ethernet and token buses.
broadcast	Sending information to all users of a particular service. An Ethernet broadcast, for example, sends an Ethernet packet to every address on the network.

---

<b>broadcast calling style</b>	A calling style in which a request PDU is sent to a user-specified set of servers. The client stub state machine can be customized to await multiple replies. [Netwise RPC Tool]
<b>broadcast router adjacency</b>	A router connected to the same broadcast circuit as this node.
<b>brouter</b>	<i>Bridge/router.</i> A device that forwards messages between networks at both network and data link levels.
<b>BSC</b>	<i>Bisynchronous.</i> See <i>Bisynch.</i>
<b>BSD</b>	<i>Berkeley Software Distribution.</i> Term used when describing different versions of the Berkeley Unix software, as in "4.3BSD Unix". [RFC 1177]
<b>BSI</b>	<i>British Standard Institute.</i>
<b>Btree</b>	See <i>binary tree.</i>
<b>Btrieve</b>	Novell programs that allow use of the Btree access method to retrieve data from servers.
<b>bucket</b>	A group of a file's virtual blocks used for I/O transfer.
<b>buffer</b>	A portion of main memory on a computer used to hold data.
<b>buffering style</b>	A user-selectable method for managing memory buffers when encoding data to or decoding data from a PDU. The RPC Tool supports two buffering styles: packet and mixed. [Netwise RPC Tool]
<b>bursty traffic</b>	Data communications term referring to an uneven pattern of data transmission.
<b>bus</b>	The part of a computer that connects peripheral devices so that they may communicate with the CPU and memory. IBM's Micro Channel Architecture is an example of a peripheral bus architecture. Also refers to any non point-to-point network with a multiple access characteristic, such as an Ethernet or token bus.
<b>C</b>	A programming language. Often used with computers running the Unix operating system.

CA	<i>See certification authority.</i>
CAB	Hosts are connected to crossbar switches via communication processor boards called CABs. The CAB presents a memory-mapped interface to user processes and off-loads all protocol processing from the host. [RFC 1152]
cable patch panel	A device located in the satellite equipment room. Used to connect two sets of wire (e.g., the wire from the satellite to the office and the wire between the satellites).
cached	A piece of information that is retained in main memory instead of being flushed to disk. Keeping information cached alleviates the need to go to the disk to retrieve the data.
CACM	<i>Communications of the ACM.</i> A technical journal.
CAD/CAM	<i>Computer aided design/computer aided manufacture.</i> Software/hardware combinations for the automation of engineering environments.
call descriptor	A data structure of the Netwise-defined type <code>call_desc</code> , available to the remote procedure which contains information concerning a specific remote procedure call request. [Netwise RPC Tool]
callback	A special case of a chained call in which the destination of the chained call is the same application process that made the parent request. [Netwise RPC Tool]
called stub	A piece of code used in an RPC mechanism. The called stub masks the remote nature of the call from the procedure on the RPC server.
calling style	Description of the operational characteristics of a remote procedure call. For example: oneshot, separable, idempotent, batched, broadcast, and no-reply. [Netwise RPC Tool]
capture	The act of removing a token from the ring.
Carrier Sense-Multiple Access/Collision Detect	The methodology used in Ethernet to mediate access to a single physical medium among multiple computers.
cartesian product	Given two lists of data, the cartesian product is the set of every possible combination of the two lists.



---

CASA	One of five gigabit testbeds in the Internet. This project involves WAN links between Los Alamos National Laboratory, Caltech, and the San Diego Supercomputer Center.
CASE	<i>See Computer-Aided Software Engineering.</i>
catenet	A network in which hosts are connected to networks with varying characteristics and the networks are interconnected by gateways (routers). The Internet is an example of a catenet. [RFC 1177]
CATV	<i>Community Antenna TV.</i> The type of cable used in broadband networks.
CBC	<i>Cypher Block Chaining.</i> A mode of the DES encryption method.
CCA	<i>Conceptual Communication Area.</i> Part of the ISO Virtual Terminal service. The CCA is an abstract area which provides a common view of the virtual terminal between two applications.
CCIRN	<i>Coordinating Committee for Intercontinental Research Networks.</i> A committee that includes the United States FNC and its counterparts in North America and Europe. Co-chaired by the executive directors of the FNC and the European Association of Research Networks (RARE), the CCIRN provides a forum for cooperative planning among the principal North American and European research networking bodies. [RFC 160]
CCITT	<i>Comité Consultatif International Télégraphique et Téléphonique</i> (Consultative Committee for International Telephone and Telegraph). Standards-making body administered by the International Telecommunications Union.
CCR	<i>Commitment, concurrency, and recovery.</i> Part of the OSI CASE services that allows the coordination of multiple users' access to data on multiple nodes.
CCS	<i>Common communications support.</i> A portion of IBM's System Application Architecture (SAA).
CCTA	<i>Central Computer and Telecommunications Agency.</i> Agency in the United Kingdom.

CD	<i>See common domain.</i>
CDA	<i>See Compound Document Architecture.</i>
CD-ROM	<i>Compact Disk-read only memory.</i> Optical disks that are mastered and then can only be read. Used for read-only databases.
CDT	<i>Credit.</i> A field used for flow control in the OSI transport service.
CEN	<i>Comité Européen de Normalisation.</i> European standards making body.
CENELEC	<i>Comité Européen de Normalisation Electrotechnique.</i> European standards body.
CERFnet	<i>California Education and Research Federation Network.</i>
certification authority	A network-based software process used in X.509 or RSA (public-key) authentication schemes. The certification authority maintains the public keys for users.
chained call	A nested remote procedure call (the child call) made during the handling of another remote procedure call request (the parent call). [Netwise RPC Tool]
channel	An IBM term referring to a direct high-speed connection into the 370 architecture machine. A "channel attach" device operates at speeds of up to 3 Mbps, as opposed to more traditional devices that attach to a communications controller at 56 kbps.
CHAR	<i>Character.</i>
character repertoire	Used in the OSI Virtual Terminal protocols. The set of possible characters a terminal can display.
characteristic attribute	A system management concept for an attribute that can be changed or modified by a director, thus affecting the behavior of that entity.
CHARGEN	<i>Character Generator Protocol.</i> Elective standard defined in RFC 864.
CheaperNet	Another term for ThinWire Ethernet cables.

CHIPCOM	A manufacturer of broadband Ethernet equipment.
choice expression	An augmented C language expression that indicates which variant of a union the RPC code should send across the network. [Netwise RPC Tool]
CI bus	<i>Computer-room interconnect.</i> Refers to the 70-Mbps bus and controllers used in the VAX Cluster arrangement. To be contrasted with Local Area VAX Clusters that use a 10-Mbps Ethernet as the transport mechanism.
CIC	<i>Certificate Integrity Check.</i> A certificate integrity check is a quantity used to verify that certificate contents have not been changed.
CICS	<i>Customer information control system.</i> An IBM data communications interface used typically with the MVS operating system.
CIF	<i>Caltech Intermediate Form.</i> A graphics format option in the Line Printer (lpr) protocol.
CIGOS	<i>Canadian Interest Group for Open Systems.</i>
C Input Specification Language	ANSI C with augmented syntax to accommodate the distributed environment used for writing Netwise RPC specifications. [Netwise RPC Tool]
C-ISL	<i>See C Input Specification Language.</i>
circuit	A term used in networking that refers to a logical stream of data between two users of the network. A single physical link may have several virtual circuits running on it.
clearinghouse	A collection of names, such as used in DNS. A user would query the clearinghouse (also known as a name server) to find the location of a particular resource at that time.
client	A module that uses the services of another module. The session layer is a client of the transport layer, for example.
client application process	An application process that makes remote procedure calls to be satisfied by a server application process. Synonymous with client process. [Netwise RPC Tool]

client header file	A text file generated by the Netwise RPC Compiler for the client that contains C language macro definitions. The client source code includes the client header file during compilation with a C Compiler. [Netwise RPC Tool]
client source file	A C source code file created by the RPC Compiler that includes client stub code and pack and unpack procedures. [Netwise RPC Tool]
client stub	Source code generated by the Netwise RPC Compiler that allows user-written client code to call a remote procedure as if it were a local procedure. The programmer links one or more client stubs into the client program. [Netwise RPC Tool]
CLNL	<i>Connectionless network layer.</i> OSI network layer.
CLNS	<i>Connectionless network service.</i> One of two options for the OSI network layer. <i>See also</i> CONS.
Closed User Group	Data communications concept for CCITT (X.25 and ISDN) where only certain users (network addresses) can access a local connection.
cluster	A collection of records in the MDB which together represent an element. Cluster records are defined in instance files. Also called cluster record or instance record. [SunNet Manager]. Also, a file system concept. A disk is made up of a series of clusters. The clusters are dynamically allocated to files and directories as needed. Similar to a block.
Clusters	A DEC architecture for VMS nodes only that allows several systems to have a common security/management domain and transparent access to the same disk drives. <i>See</i> VAX Cluster.
CMIP	<i>Common Management Information Protocol.</i> OSI network management protocols, similar in function to SNMP.
CMISE	<i>Common Management Information Service Element.</i> The CMIP service element that provides the basic management services. The CMISE provides both confirmed and unconfirmed services for reporting events and retrieving and manipulating management data. [RFC 1095]



<b>CMOT</b>	<i>CMIP over TCP/IP.</i> An implementation of the OSI CMIP protocols over TCP/IP instead of OSI. Allows current networks to take advantage of the enhanced capabilities of CMIP without running the entire OSI protocol stack.
<b>CMS</b>	<i>Code Management System or Conversational Monitor System.</i> Code Management System is a DEC software product used in the VMS environment as a library for program development. Conversational Monitor System is the user interface on IBM's VM/CMS operating system.
<b>CMT</b>	<i>Connection management.</i> Abbreviation used in the ISO FDDI standard.
<b>CMU</b>	<i>Carnegie Mellon University.</i>
<b>CN</b>	<i>Common Name.</i> Abbreviation used for X.400 addresses.
<b>CNRI</b>	<i>Corporation for National Research Initiatives.</i>
<b>coax</b>	<i>Coaxial.</i> A type of cable, used for IBM 3270 terminals, as well as for baseband and ThinWire Ethernets.
<b>COBOL</b>	<i>Common business-oriented language.</i> One of the first standardized computing languages. <i>See CODASYL.</i>
<b>CODASYL</b>	<i>Conference on Data Systems Languages.</i> The folks that brought you COBOL as well as the CODASYL standard for databases using the network model of data management.
<b>common domain</b>	An administrative domain which is not a member of a higher-level domain. A common domain is the highest level in the routing hierarchy. There is no single domain above the common domain. In this sense, the routing hierarchy is in fact multiple hierarchies, with the common domain as the highest element of each hierarchy. [RFC 1136]
<b>Communication and Control Services</b>	NAS concept for the API that allows an application to access communication services on the network.
<b>component</b>	A type of network element corresponding to devices like computers, operating systems, gateways, etc. [SunNet Manager]
<b>Compound Document</b>	A document with multiple types of information, such as text, graphics, voice.

Compound Document Architecture	DEC architecture for the creation, storage, and manipulation of compound documents.
Computer-Aided Software Engineering	Software used to design software. Source code management, data dictionaries, and entity-relationship diagramming tools are all examples.
concentrator	A node on an FDDI ring which provides connections to additional stations. Known as a multistation access unit (MAU) in 802 token rings.
concurrency	When multiple users attempt to access the same resource. A lock manager addresses the problem of maintaining the integrity of resources in a concurrent environment.
conferencing	A term used for communication software that allows participants to "post" notes. Contrasts with electronic mail in that participants do not have to be explicitly addressed. Also known as a bulletin board.
config.sys	An MS-DOS file read in at boot time which contains the names of additional device drivers, such as a driver for extended memory or a peripheral such as a scanner.
congestion	Too much traffic for a given circuit.
connection	A point-to-point network link connecting two other elements. These come in two types, regular and simple. Regular connections are full-fledged objects that can have properties and can be managed by agents. Simple connections are a user convenience feature that have a graphical representation but no database representations (they don't have properties and can't be managed by agents). [SunNet Manager]
connection manager	A cluster term used to refer to the software component in a VAX Cluster or LAVC that maintains the integrity of the cluster by managing state transitions.
CONS	<i>Connection Oriented Network Service.</i>
console	The window-based manager application that comes with SunNet Manager. [SunNet Manager]
CONTENT	<i>Content Type Header Field.</i> Recommended standard defined in RFC 1049.

---

control	The ability to modify object attribute values to change the characteristics of managed objects. [SunNet Manager]
cookie	An opaque piece of data used in NFS to hold the file handle sent by the server to the client.
coprocessors	An add-on board or chip on a computer that supplements the services of the main CPU.
core gateway	Historically, one of a set of gateways (routers) operated by the Internet Network Operations Center at BBN. The core gateway system forms a central part of Internet routing in that all groups must advertise paths to their networks from a core gateway. [RFC 1177]
COS	<i>Corporation for Open Systems.</i>
COSAC	<i>Canadian Open Systems Application Criteria.</i>
CPE	<i>Customer Premises Equipment.</i>
CPP	<i>See cable patch panel.</i>
CPSR	<i>Computer Professionals for Social Responsibility.</i>
CPU	<i>Central processing unit.</i> You know—the computer part of the computer.
CR	<i>Connection Request.</i> A protocol data unit type in the OSI transport service.
CRC	<i>See cyclic redundancy check.</i>
credit	A flow control mechanism used in network protocols. A node is allowed to send a packet only if it has a credit available. If not, it must wait for the remote node to send one.
CREN	<i>The Corporation for Research and Educational Networking.</i> BITNET and CSNET have recently merged to form CREN. [RFC 1177]
CRLF	<i>Carriage Return/Line Feed.</i>
CRMA	<i>Cyclic-Reservation Multiple-Access.</i> An access method used in very high-speed networks.
CRT	<i>Cathode ray tube.</i>

CS	<i>Checksum or Computer Science.</i> A checksum is a method of assuring the integrity of data transmitted.
CSA	<i>Client services agent.</i> Part of the Manufacturing Automation Protocol (MAP) Directory Service.
CSMA/CD	<i>See carrier sense-multiple access/collision detect.</i>
CSNET	<i>Computer + Science Network.</i> A large data communications network for institutions doing research in computer science. It uses several different protocols including some of its own. CSNET sites include universities, research laboratories, and commercial companies. [RFC 1177]
CTERM	<i>Communications Terminal Protocol.</i> Part of the virtual terminal service in layer 6 of the Digital Network Architecture. An alternative to the CTERM services is the Local Area Transport Architecture (LAT).
CTM	<i>Current transformation matrix.</i> A term used in the PostScript Imaging System to denote the transformation between user space and device space.
CUA	<i>Common user access.</i> The common user interface component of IBM's System Application Architecture. Similar concepts exist with Xerox's Open Look standard and within the DECwindows environment.
CUDF	<i>Call User Data Field.</i> Field in an X.25 header.
CUG	<i>See Closed User Group.</i>
Cullinet	Software company that markets the IDMS database package.
Culprit	A Cullinet software package used in the IBM environment that is used to extract data from another DBMS such as IMS.
custom instructions	User-written instructions that are inserted into the client stubs, server stubs, or dispatcher by adding Hooks, Entry, Exit, or Traps statements to the Netwise RPC specification. [Netwise RPC Tool]
cyclic redundancy check	A number derived from a set of data that will be transmitted. By recalculating the CRC at the remote end and comparing it to the value originally transmitted, the receiving node can detect some types of transmission errors.



---

DA	<i>Destination address.</i>
daemon	A Unix term referring to a process that is not connected with a user but performs services, such as a mail daemon. The equivalent VMS term is a detached process.
DAP	<i>Division Advisory Panel or see Data Access Protocol or Directory Access Protocol.</i> The Division Advisory Panel is an advisory group to the National Science Foundation Division of Networking and Communication Research and Infrastructure.
DARPA	<i>Defense Advanced Research Projects Agency.</i> A Department of Defense agency that has helped fund many computer projects including ARPANET, the Berkeley version of Unix and TCP/IP.
DAS	<i>See Dual Attachment Station.</i>
Data Access Protocol	A protocol used in the Digital Network Architecture in layer 6. Provides a rich set of functions used for exchanging data between two nodes of the network. <i>See also File Access Listener.</i>
data country code	An ISO-administered format for unique OSI addresses based on geographic location. The codes are defined in ISO 3166. <i>See also International Code Designator.</i>
data report	Object attributes reported for analysis, as opposed to event report. [SunNet Manager]
Data terminal equipment	An X.25 term referring to the interface to users' equipment as opposed to the DCE interface to the network.
datagram	The unit transmitted between a pair of internet modules. The Internet Protocol provides for transmitting blocks of data, called datagrams, from sources to destinations. The Internet Protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control. <i>See IP.</i> [RFC 1177]
DATAPAC	Canadian public packet switched network.
DAYTIME	<i>Daytime Protocol.</i> Elective protocol defined in RFC 867.

DB2	An IBM database package based on the relational model and the SQL query language.
DBMS	<i>Database Management System.</i> Software that allows the centralized storage of data with multiple concurrent users, access control, and the use of a high-level data manipulation language such as SQL.
DC	<i>Disconnect Confirm.</i> A protocol data unit type in the OSI transport service.
DCA	<i>Document Content Architecture or Defense Communications Agency.</i> Document Content Architecture is an IBM architecture similar in function to DEC's Compound Document Architecture (CDA). The Defense Communication Agency is responsible for the Defense Data Network.
DCC	<i>See data country code.</i>
DCE	<i>Data circuit-terminating equipment.</i> Term used in X.25 networks for the device on the edge of the network that accepts and initiates calls. The DCE is in turn connected to a DTE which communicates with the user.
DCL	<i>Digital command language.</i> The user interface in the VMS operating system. Similar to the C shell in the Unix operating system.
DD	<i>Domain Defined.</i> A locally-defined attribute for X.400 addresses.
DDN	<i>Defense Data Network.</i> A network for the Department of Defense and their contractors based on the TCP/IP and X.25 networking protocols.
DDN NIC	The network information center at SRI International. It is the primary repository for RFCs and Internet drafts, as well as providing other services. [RFC 1177]
DDP	<i>Datagram Delivery Protocol.</i> AppleTalk network-level protocol.

DDS	<i>See Distributed Directory Service.</i>
deadlock	A term used in a concurrent environment. When one user holds a lock on a resource and is waiting for another resource to free up and a second user has the reverse situation. The deadlock must be broken by arbitrarily picking one of the users and releasing its current lock. The deadlock is also known as a deadly embrace.
DEC	<i>Digital Equipment Corporation.</i>
DECconnect	A DEC cabling architecture used for facilities wiring.
DECmcc	<i>DEC Management Control Center.</i> DEC software which is the implementation of the EMA management architecture.
DECnet	An implementation of the Digital Network Architecture by DEC, as opposed to implementations of DNA by other vendors.
DECwindows	DEC's implementation of the MIT X Window System.
default route	A routing table entry which is used to direct any data addressed to any network numbers not explicitly listed in the routing table. [RFC 1177]
DEK	<i>Data Encryption Key.</i> Used for encryption of message text and (with certain choices among a set of alternative algorithms) for computation of message integrity check (MIC) quantities. DEKs are generated individually for each transmitted message; no predistribution of DEKs is needed to support privacy-enhanced message transmission. [RFC 1113]
DELNI	<i>DEC Local Network Interconnect.</i> "Ethernet in a Can." A multiport transceiver made by DEC.
Delta T	<i>Delta time.</i> Sniffer Analyzer indication of time elapsed between consecutive packets on the network. Contrast to relative time, which is the time that has elapsed since a particular anchor packet was sent.
DES	<i>Data Encryption Standard.</i> One type of encryption scheme.

designated router	A DNA or OSPF routing concept. A given broadcast circuit (such as an Ethernet) will have a designated router, which is used by end nodes to forward all packets which will need routing decisions.
DF	<i>Don't fragment.</i> An IP flag indicating that this packet should not be fragmented.
DFS	<i>Distributed File Service (or System).</i> A DEC product similar to the Network File System (NFS). Both allow remote files to appear as though they were locally mounted on a workstation, allowing diskless nodes. DFS uses the DNS name server.
DIB	<i>See Directory Information Base.</i>
Digital Network Architecture	DEC architectures for networking. DECnet is an implementation of DNA.
director	The management entity that issues directives, commands to agents to perform a function. SunNet Manager is an example of a director.
Directory Access Protocol	X.500 protocol used for communication between a Directory User Agent and a Directory System Agent.
Directory Information Base	X.500 term for the distributed database used to keep track of application entities, people, terminals, distribution lists, or other objects.
Directory Information Tree	X.500 term for the hierarchical name space of objects contained in the Directory Information Base.
Directory Management Domain	An administrative group responsible for managing a portion of the directory information base. A directory management domain contains at least one directory system agent and zero or more directory user agents.
Directory System Agent	X.500 term for the software that maintains and provides access to the Directory Information Base.
Directory System Protocol	X.500 protocol used for communication between directory system agents.



Directory User Agent	X.500 term for the software that provides services to the user. The Directory User Agent uses the OSI ROSE protocols to communicate with one or more Directory System Agents.
DIS	<i>Draft Information Standard.</i> The step before becoming a formal international standard in the ISO process. At this point the standard is considered to be technically correct and only minor corrections are anticipated.
DISC	<i>Disconnect.</i>
DISCARD	<i>Discard Protocol.</i> Elective protocol defined in RFC 863.
discriminated union	A discriminated union is a type composed of a discriminant followed by a type selected from a set of prearranged types according to the value of the discriminant. The type of discriminant is either integer, unsigned integer, or an enumerated type, such as boolean. The component types are called "arms" of the union, and are preceded by the value of the discriminant which implies their encoding. [RFC 1014]
DISOSS	<i>Distributed Office Support System.</i> An IBM product that serves as a distributed library of documents. <i>See also DCA.</i>
dispatch function	Agent routine that calls an appropriate agent function to service the request. [SunNet Manager]
dispatcher	A procedure generated by the Netwise RPC Compiler as a component of the server source code. The dispatcher receives requests from client procedures and directs the requests to the appropriate server stubs. [Netwise RPC Tool]
distinguished name	An X.500 concept for the unique name of an object derived from its location in the Directory Information Tree.
distributed application	An application whose code is executed by two or more application processes on a network. [Netwise RPC Tool]
distributed database	Looks to the user like a single database but is in fact a collection of several different data repositories.
Distributed File Service	DEC product to make files on the network all appear local. Similar to the Network File System.
distributed naming service	Network-based service to allow a user to find the current address of a given resource, such as a printer or file system.

DIT	<i>See Directory Information Tree.</i>
DL	<i>Distribution List.</i> Abbreviation used for X.400 addresses.
DLAL	<i>Dual letter acronym listing.</i> <i>See also MLAL.</i>
DLC	<i>Data Link Control.</i> Sniffer Analyzer notation for data link layer information.
DMA	<i>Direct memory access.</i> Allows a device on a computer to access main memory without a CPU interrupt.
DMD	<i>See Directory Management Domain.</i>
DMF-MAIL	<i>Digest Message Format for Mail.</i> Elective Internet protocol defined in RFC 1153.
DML	<i>Data manipulation language.</i> A language, such as SQL, used for retrieving and manipulating data in a database system.
DNA	<i>See Digital Network Architecture.</i>
DNA Naming Service	A distributed naming service used heavily in DNA Phase V. Known as DNANS to differentiate it from the Domain Name System.
DNS	<i>See DNA Naming Service or Domain Name System.</i>
domain	An area within which a particular service is performed. In a messaging domain, a message transfer agent for that domain is able to deliver a message.
Domain Name System	The Domain Name System is a mechanism used in the Internet for translating names of host computers into addresses. The DNS also allows host computers not directly on the Internet to have registered names in the same style. [RFC 1177]
Domain Specific Part	The part of the OSI address that is locally assigned.
DOS	<i>Digital Operating System.</i> Microsoft operating system for IBM/PCs.
DOS/VSE	<i>Digital Operating System/Virtual Storage Extended.</i> An IBM operating system used on 370 architecture mainframes.

dpi	<i>Dots per inch.</i> A measure of the resolution of printers or scanners.
DR	<i>Disconnect Request.</i> A protocol data unit type in the OSI transport service.
Draft Standard Protocol	Classification for Internet standards. The IAB is actively considering this protocol as a possible Standard Protocol. Substantial and widespread testing and comment are desired. Comments and test results should be submitted to the IAB. There is a possibility that changes will be made in a Draft Standard Protocol before it becomes a Standard Protocol. [RFC 1140]
DRP	<i>Digital Routing Protocol.</i>
DSA	<i>See Directory System Agent.</i>
DSAP	<i>Destination service access point.</i> The address for the destination user of a service. A remote IPX process would be considered the DSAP from the point of view of the local data link module.
DSE	<i>Directory Services Element.</i> X.500 service provider.
DSP	<i>See Domain Specific Part .</i>
DST	<i>Destination address.</i> Sniffer Network Analyzer abbreviation.
DTE	<i>See Data terminal equipment.</i>
DU	<i>Data unit.</i> Part of the OSI File Transfer Access and Management (FTAM) protocols.
DUA	<i>See Directory User Agent.</i>
Dual Attachment Station	FDDI term for a node that is attached to both the primary and secondary fiber optic cables (as opposed to a node that is connected to the ring via a concentrator).
dual-porting	Making a disk drive available to two different computers.
DVI	<i>Device Independent.</i> The intermediate output format used in the Tex text processing system.
E.163	CCITT numbering scheme for public switched telephone networks.

E.164	CCITT standard for numbering in an ISDN environment.
EA	<i>Expedited Acknowledge.</i> A protocol data unit type in the OSI transport service.
EARN	<i>European Academic Research Network.</i>
Easynet	DEC's internal communications network.
EB	<i>Encapsulation Boundary.</i> A convention used to identify the end of an encapsulated message. Often consists of a line of dashes. Documented in RFC 934.
EBCDIC	<i>Extended Binary Coded Decimal Interchange Code.</i> A character code scheme used in IBM environments. <i>See also ASCII.</i>
EBNF	<i>Extended Backus Naur Form.</i> An extended version of Backus Naur Form as defined in Section 2 of RFC 822.
ECB	<i>Electronic Codebook.</i> A mode of the DES encryption method.
Echo	A maintenance protocol in XNS. Used to echo information back across the network and thus test the connection. Also a TCP option that has the remote site echo the received data using an "echo reply" option. Used in timing the round-trip transmission (RTT).
ECL	<i>See End Communication Layer.</i>
ECMA	<i>European Computer Manufacturers Association.</i> A leading European standards body.
ED	<i>Expedited Data.</i> A protocol data unit type in the OSI transport service.
EDIFACT	<i>EDI for Administration, Commerce, and Trade.</i> Emerging international Electronic Data Interchange (EDI) standard.
EGP	<i>External Gateway Protocol.</i> A protocol which distributes routing information to the routers and gateways which interconnect networks. [RFC 1177]
EHF-MAIL	<i>Encoding Header Field for Mail.</i> Elective Internet protocol defined in RFC 1154.



EI	<i>Entity Identifier.</i> A subfield used in Privacy Enhanced Mail to indicate the entity (user) to which a particular message or certificate applies.
EIA	<i>Electronic Industries Association.</i> Trade association.
EIA-423	Standard interface definition for serial devices.
Elective Protocol	Protocol status for Internet standards. A system may or may not implement an elective protocol. The general notion is that if you are going to do something like this, you must do exactly this. [RFC 1140]
electronic mail	Collection of programs that allow users to exchange messages across a network.
element	An individual object subject to management. Examples are "the machine called mgrhost," "ethernet interface ie0," and "building 14." [SunNet Manager]
element category	A family of element types. The SunNet Manager Console defines four element categories: bus, component, connection, and view. [SunNet Manager]
element type	A structural definition of a particular kind of element. Examples are component.sun3 and bus.ethernet. [SunNet Manager]
EMA	<i>See Enterprise Management Architecture.</i>
EMACS	An extensible editor developed at MIT based on the LISP programming language.
Email	<i>Electronic mail.</i> Software and networks that allow the exchange of messages between users.
EMUG	<i>European MAP Users Group.</i>
End System	An OSI system on which applications run. An End System has full seven-layer OSI functionality. Basically equivalent to an Internet host. [RFC 1136]
endian	How a computer stores a multi-octet piece of data (e.g., a four-byte integer). <i>See big endian and little endian.</i>

endpoint	<i>See binding endpoint.</i>
Enterprise Management Architecture	A DEC architecture for network management user interfaces that can work with multiple displays and protocols. DEC-mcc is the implementation of this architecture.
entity	Network management term for a manageable piece of a distributed system.
Entry statement	A C language augmentation that instructs the Netwise RPC Compiler to add custom instructions to the source code generated for the designated state machine. [Netwise RPC Tool]
enumerator	Concept used in AppleTalk Name Binding Protocol searches. The enumerator value is used to distinguish among several instances of an entity on a single socket.
EOF	<i>End of file.</i> A mark that tells the file system that it has reached the end of a file.
EON	An acronym for Experimental OSI Network, a name for the proposed experimental OSI-based internetwork that uses the IP over the Internet as a subnetwork. [RFC 1070]
EON-UDP	A name for the proposed experimental OSI-based internetwork that uses the UDP/IP over the Internet as a subnetwork. [RFC 1070]
EOR	<i>End of record.</i>
EPHOS	<i>European Procurement Handbook for Open Systems.</i>
EPS	<i>Encapsulated PostScript.</i> The EPS file format is two things: first, a specific file representation that allows both a PostScript program and a low-resolution bit image to reside in the same file; second, it is a set of guidelines governing the PostScript code itself. The primary purpose of this file format is to facilitate the preparation of illustrations that may be included into document formatting systems in a regular way. [Compuserve Posting by Glen Reid, Adobe Systems]
EPSF	<i>EPS File Format.</i>
ER	<i>Error.</i> A protocol data unit type in the OSI transport service.
Error	XNS protocol used to report errors across a network.

ES	End system as defined by OSI: an OSI network layer entity that provides the OSI network layer service to a transport layer. [RFC 1070]
ESC	<i>Escape.</i>
ESH PDU	<i>End System Hello PDU.</i> From OSI Network Service Definition.
ESIS	<i>End System to Intermediate System.</i> Protocol defined in ISO 9542 to allow end systems and intermediate systems on the same subnetwork to communicate.
EST	<i>Eastern Standard Time.</i>
Ethernet	A data link protocol jointly developed by Intel, Xerox, and DEC and subsequently adopted by the IEEE as the 802.3 standard. Several upper-layer protocols, including DECnet, TCP/IP, and XNS, use the Ethernet as an underlying transport mechanism. Ethernet is to be contrasted with other data link protocols such as the token ring, DDCMP, and SDLC.
Ethernet controller	A device controller that gives the computer access to the Ethernet services. Typically, the CSMA/CD protocols are built into the controller so the CPU doesn't have to worry about the details of the protocol.
Ethernet Version 2.0	The second version of the original specification for Ethernet, which differs slightly from the IEEE 802.3 standard.
Ethertype	Field in Version 2.0 of Ethernet that indicates the type of user (DECnet, NetWare, or TCP/IP, for example).
EUnet	<i>European Unix Network.</i> Network originally based on UUCP.
EurOSInet	<i>European OSI Promotional Group.</i> OSI Interoperability testing network for Europe.
EVD	<i>See event dispatcher.</i>
event	The occurrence of a particular change in the state of an object attribute. [SunNet Manager]

event dispatcher	A process that acts as a rendezvous for event reports. Management applications can ask this daemon to send them selected copies of reports based on particular criteria. [SunNet Manager]
event log	A record of significant events kept in the director.
event report	The notification an agent sends when an event is detected on a managed object. [SunNet Manager]
event sink	The destination in the network where significant events should be sent. Examples of a sink are a file, a program, and a terminal.
EWOS	<i>European Workshop for Open Systems.</i>
EXEC	<i>Executable.</i>
executable image	A program that is ready to run on an operating system. A program starts as source code and gets compiled to generate object code. The object code is then linked to form an executable image.
Exit statement	A C language augmentation that instructs the Netwise RPC Compiler to add custom instructions to the source code generated for the client stubs, server stubs, or dispatcher, to be executed after entering the designated state machine. [Netwise RPC Tool]
experimental protocol	Classification for Internet standards. Experimental protocols are those that are developed as part of an ongoing research project not related to an operational service offering. While they may be proposed as a service protocol at a later stage, designation of a protocol as experimental may sometimes be meant to suggest that the protocol, although perhaps mature, is not intended for operational use. [RFC 1140]
expression macro	A C language augmentation accepted by the Netwise RPC Compiler that consists of two categories: pointer macros and index macros. Expression macros may be used in choice expressions and termination expressions. <i>See RPC macro.</i> [Netwise RPC Tool]
External Data Representation	Presentation layer protocol developed by Sun Microsystems as part of NFS.



<b>external variable mode</b>	A C language augmentation that qualifies how values in an external variable are to be used by a remote procedure. One of in, out, or in out. [Netwise RPC Tool]
<b>F.60</b>	CCITT standard for telex services.
<b>F.69</b>	CCITT standard for telex addresses.
<b>F.110</b>	CCITT standard for maritime mobile service.
<b>F.160</b>	CCITT standard for international public facsimile services.
<b>F.200</b>	CCITT standard for teletex services.
<b>F.201</b>	CCITT standard for internetwork teletex and telex services.
<b>F.300</b>	A set of CCITT recommendations for Videotex systems.
<b>F.401</b>	CCITT standard for the naming and addressing for public message handling services.
<b>F.410</b>	CCITT standard for the public message transfer service.
<b>F.415</b>	CCITT standard for intercommunication with public physical delivery services.
<b>F.420</b>	CCITT standard for the public interpersonal messaging service.
<b>F.421</b>	CCITT standard for communication between the X.400 interpersonal messaging service and the telex service.
<b>F.422</b>	CCITT standard for communication between the X.400 interpersonal messaging service and the teletex service.
<b>F.500</b>	CCITT standard for international public directory services.
<b>FADU</b>	<i>File access data unit.</i> The unit of access in the OSI FTAM service.
<b>FAL</b>	<i>See File Access Listener.</i>
<b>fault tolerance</b>	An attribute of a computer system that reflects its degree of tolerance to hardware and software failures while continuing to run.

fax	<i>Facsimile.</i> A messaging service based on transmitting bit maps of 200 dots per inch across dial-up telephone lines.
FC	<i>Frame control.</i> Token ring field that indicates whether the packet is a MAC-layer management packet (e.g., a token, beacon, AMP, or SMP) or is carrying LLC data.
FCCSET	<i>Federal Coordinating Council for Science, Engineering and Technology.</i>
FCS	<i>Frame check sequence.</i> A mechanism like a CRC or checksum to guarantee the integrity of a packet of data.
FDDI	<i>See Fiber Distributed Data Interface.</i>
FEC	<i>Forward Error Correction.</i> FEC is a useful approach when the bandwidth/delay ratio of the physical medium is high, as can be expected in transcontinental photonic links. A degenerate form of FEC is to simply transmit multiple copies of the data; this allows one to trade bandwidth for delay and reliability, without requiring much intelligence. [RFC 1077]
FFFF	<i>15 15 15 15.</i> A hexadecimal number. Hexadecimal is base 16 numbering in which the symbols A through F are used to represent the digits 10 through 15. FFFF is equivalent to a 32 bit quantity with all bits set.
FFT	<i>Final form text.</i> A version of IBM's Document Content Architecture (DCA).
FHSIZE	<i>File Handle Size.</i>
Fiber Distributed Data Interface	A 100-Mbps fiber optic LAN standard based on the token ring.
field	A term used in designing forms-based systems such as database applications. A field is a portion of the screen used for data input which is automatically mapped to a variable. The field may have attributes such as reverse video or default values.
FLFMD	<i>Function Interpreters for Function Management Data.</i> IBM's presentation layer protocol for SNA.
FIFO	<i>First In, First Out.</i> A queue management technique.

<b>File Access Listener</b>	A process invoked across the network by a user trying to access data on remote systems using DEC's Data Access Protocol. A FAL is a remote DAP-speaking process invoked by the Record Management Services on the local node.
<b>file cache</b>	Keeping portions of files in main memory. When the computer requests data and it is in the cache, it alleviates the need to refetch it from the much slower disk drive.
<b>file system</b>	The portion of an operating system that is responsible for storing and retrieving pages of data onto a disk.
<b>finder</b>	<i>See Macintosh Finder.</i>
<b>finger</b>	<i>Finger Protocol.</i> Elective Internet protocol defined in RFC 742.
<b>finite state machine</b>	A way of describing a network module as a set of states which, based on inputs and conditions, perform transitions to other states and outputs.
<b>FIPS</b>	<i>Federal Information Processing Standard.</i>
<b>floating point</b>	A native data type on most operating systems. A floating point number is one that can have numbers after the decimal point, in contrast to an integer, which cannot.
<b>FNC</b>	<i>Federal Networking Council.</i> Coordinating group for United States government networking.
<b>FOO</b>	A common variable used in examples. Derived from the military term FOOBAR (F*d Up Beyond All Recognition).
<b>Form Interface Management System</b>	Draft ISO standard for forms management.
<b>fourth-generation language</b>	A group of new languages often linked with database packages such as Ingres or Oracle. In contrast with Fortran and other third-generation languages.
<b>frame</b>	A series of bytes of data encapsulated with a header. The data link layer sends frames of data back and forth. "Frame" is often used interchangeably with "packet," although technically a packet refers to data from the network layer of the protocol stack. A packet is thus usually contained inside a frame.

FRICC	<i>Federal Research Internet Coordinating Committee.</i> An informal group since replaced by the Federal Network Council.
front end	A program with which a user interacts. The front end sends off requests to the back end for data.
FS	<i>Frame status.</i> Token ring field that indicates if the address on a packet has been recognized by a station and if the data has been copied.
FSM	<i>Finite State Machine.</i>
FTAM	<i>File Transfer, Access, and Management.</i> The OSI application layer service that provides access to virtual file stores on foreign systems. Similar to the DNA DAP protocols in purpose.
FTP	<i>File Transfer Protocol.</i> The Internet standard high-level protocol for transferring files from one computer to another. [RFC 1177]
full name	A unique, unambiguous name in the name space.
full-duplex	A data communications term that indicates that both ends of a communications link can transmit simultaneously. Contrasted with half-duplex where only one side can transmit at one time.
function	Takes a piece of data as input and returns a value. For example, the query language has functions that can accept a date and return the day of the week on which the date falls.
FYI	<i>For your information.</i>
Gandalf	A manufacturer of communications equipment including data switches.
gateway	There are two somewhat conflicting definitions of gateway, both used in networking. In the general sense, a gateway is a computer that connects two different networks together. Usually, this means two different kinds of networks such as SNA and DECnet. In TCP/IP terminology, however, a gateway connects two separately administered subnetworks, which may or may not be running the same networking protocols.



<b>GB</b>	<i>Gigabit backbone or gigabyte.</i> The gigabit backbone is an effort to increase the speed of the Internet backbone to 1 Gbps. A gigabyte is a billion bytes of data.
<b>Gb</b>	<i>Gigabit.</i> $2^{30}$ bits of information.
<b>Gbyte</b>	<i>Gigabyte.</i> One billion bytes of data.
<b>GFLOPS</b>	<i>Billion Floating Operations Per Second.</i>
<b>GGP</b>	<i>Gateway Gateway Protocol.</i> Obsolete protocol defined in RFC 823.
<b>GID</b>	<i>Group identification.</i>
<b>gigabyte</b>	<i>Billion bytes of data.</i>
<b>GKS</b>	<i>See Graphical Kernel System.</i>
<b>GKS-3D</b>	<i>Graphical Kernel System for Three Dimensions.</i> 3D extensions to the GKS standards.
<b>global entity</b>	The top level of the entity tree for management purposes. A node is an example of a global entity, with subentities including modules such as routing, MOP, or LAT.
<b>glyph</b>	A pictorial representation. The SunNet Manager Console provides three forms of glyph: the fixed-size image (components, views, and connections without both end elements in the current view), the variable-length vector with grab handles (buses), and the variable-length vector (connections with both end elements in the current view). [SunNet Manager]
<b>GM</b>	<i>General Motors.</i> Primary supporter of the Manufacturing Automation Protocol (MAP). They also make cars.
<b>GMD</b>	<i>Gesellschaft fur Mathematik und Datenverarbeitung.</i>
<b>GNA</b>	<i>Global Network Addressing.</i> A bit in the HYPERchannel header that indicates that a full 32-bit address plus domain and network numbers are present.
<b>GNU</b>	<i>Gnu's Not Unix.</i> Unix-compatible software developed by the Free Software Foundation. WGNU is the public radio station in Boulder, Colorado.

GOSIP	<i>Government OSI Protocols.</i> United States government version of the international OSI standards.
GPO	<i>Government Printing Office.</i>
GQ	<i>Generation Qualifier.</i> An abbreviation used in X.400 addresses to indicate qualifiers such as "Junior."
granularity	A term used in lock managers on an operating system. When the lock manager locks an entire file, this is locking with course granularity. When the lock manager locks a single record, this is fine granularity. Granularity is one of the factors that influence the performance of a particular application, such as a DBMS.
Graphical Kernel System	ISO standard for the creation and manipulation of 2D objects.
group	A collection of related attributes. Defined by the agent writer and specified in the agent schema. [SunNet Manager]
GRPC	<i>Generalized RPC.</i> RPC mechanism in which code, data, and execution might be variously local or remote from the procedure initiator.
half-duplex	<i>See full-duplex.</i>
handle	A number used to refer to a file or directory that is being remotely accessed on the network.
HASP	<i>Houston Automatic Spooling Program.</i> One of the original implementations of the remote job entry function on IBM equipment. Still used as a common denominator, in conjunction with Bisynch, for RJE functions.
HDTV	<i>High Definition Television.</i>
header	The portion of a packet, preceding the actual data, containing source and destination addresses and error-checking fields. [RFC 1177]
heap	A storage structure for data where data is not placed in any particular order, requiring a scan of the entire table for every retrieval.
HELLO	An interior gateway protocol.

<b>HEMS</b>	<i>High-Level Entity Mangement Systems.</i> An early set of definitions for the network management and the MIB defined in RFC 1024.
<b>heterogeneous</b>	Different.
<b>heterogeneous network</b>	A network consisting of different network protocols or kinds of computers. A network combining SNA and DNA protocols using an SNA gateway to connect the two is a heterogeneous network.
<b>hierarchical database</b>	A database that structures data as a hierarchy instead of in tables. Programmers then navigate the hierarchy to retrieve a particular row of data. IMS is an example of a hierarchical database system.
<b>hierarchical routing</b>	Routing based on domains. Interdomain routers are responsible only for getting data to the right domain. There, an intradomain router takes responsibility for routing within the domain.
<b>Hierarchical Storage Controller</b>	Stand-alone disk and tape controller used in clusters using the CI bus. The HSC is actually a modified PDP computer that has been optimized as a mass storage controller.
<b>HIPPI</b>	<i>High Performance Parallel Interface</i> An emerging ANSI standard which extends the computer bus over fairly short distances at speeds of 800 and 1600 Mbps. HIPPI is often used in a computer room to connect a supercomputer to routers, frame buffers, mass-storage peripherals, and other computers.
<b>Historic Protocol</b>	Classification for TCP/IP standards. These are protocols that are unlikely to ever become standards in the Internet either because they have been superseded by later developments or due to lack of interest. [RFC 1140]
<b>homogeneous</b>	The same.
<b>hooks</b>	Programming technique. Allows a programmer to add new code to an existing program. The existing program has hooks that execute any additional code.
<b>Hooks statement</b>	A C language augmentation that instructs the Netwise RPC Compiler to add custom instructions to the state-machine source code generated for the client stubs, server stubs, or dispatcher. [Netwise RPC Tool]

hop	A term used in routing. A hop is one data link. A path to the final destination on a net is a series of hops away from the origin. Each hop has a cost associated with it, allowing the calculation of the least cost path.
host	A computer that provides services directly to users (as opposed to a behind-the-scenes dedicated server).
Host Field	The bit field in an Internet address used for denoting a specific host. [RFC 950]
host number	The part of an internet address that designates which node on the (sub)network is being addressed. [RFC 1177]
HSC	<i>See Hierarchical Storage Controller.</i>
HSI	<i>High-Speed Serial Interface.</i> Communications controller for Sun servers that operates at up to T1 speeds (1.544 Mbps). <i>See also MCP.</i>
HT	<i>Horizontal Tab.</i>
hub	A device connected to several other devices. In ARCnet, a hub is used to connect several computers together. In a message handling service, a hub is used for the transfer of messages across the network.
HYPER integer	<i>A 64-bit integer.</i>
HYPERchannel	Data link made by Network Systems Corporation. Often used in supercomputer centers.
I.120	CCITT description of ISDN.
I.230	Description of ISDN bearer services.
I.240	Description of ISDN teleservices.
IA	<i>Issuing Authority.</i> A concept in Privacy Enhanced Mail which indicates which group is vouching for the integrity of a given certificate.
IAB	<i>Internet Activities Board.</i> The IAB is the coordinating committee for Internet design, engineering, and management. [RFC 1177]



---

IANA	<i>Internet Assigned Numbers Authority.</i>
IBM	<i>International Business Machines Corporation.</i> Big.
IBM 8228 Multistation Access Unit	A MAU sold by IBM.
IBM PC LAN	IBM PC network using broadband.
IBM Token-Ring	IBM token ring network.
IBM Token-Ring Adapter	Token ring controller card sold by IBM.
ICCB	<i>Internet Configuration Control Board.</i> An informal group replaced by the Internet Activities Board.
ICD	<i>See International Code Designator.</i>
ICMP	<i>Internet Control Message Protocol.</i> Protocol used by the IP layer of TCP/IP for exchanging routing control messages.
icon	A small pictorial object on a workstation used to represent a closed window. The user points to the icon, clicks the mouse button, and a window opens.
idempotent	Describes an operation that will yield the same result if it is executed more than once. Change X to 25 is idempotent. Add 25 to X is not.
idempotent calling style	A calling style in which the client stub state machine retries the request until it receives a reply. Note that as a result the remote procedure may be executed more than once. [Netwise RPC Tool]
IDI	<i>Initial domain identifier.</i> Part of an OSI address. Goes after the authority and format identifiers. For example, the IDI for a telephone address might be the country code.
IDMS	Cullinet's database management system for IBM systems.

IDP	<i>See Internetwork Datagram Protocol.</i>
IEC	<i>Inter-exchange carrier.</i>
IEEE	<i>Institute of Electronic and Electrical Engineers.</i> A leading standard-making body in the United States, responsible for the 802 standards for local area networks.
IEEE 488	IEEE standard for real-time data acquisition.
IEN	<i>Internet Engineering Note.</i> Predecessor to the RFC series.
IESG	<i>Internet Engineering Task Force Steering Group.</i> The governing body of the IETF.
IETF	<i>Internet Engineering Task Force.</i> A volunteer group that helps develop new technology for use in the Internet.
IGMP	<i>Internet Group Multicast Protocol.</i> Recommended Internet protocol defined in RFC 1112.
IGP	<i>Interior Gateway Protocol.</i> A protocol such as RIP used within an administrative domain. Contrast to EGPs, which are used to exchange information between administrative domains.
IHL	<i>Internet Header Length.</i> Indicates the length of the Internet (IP) header.
IK	<i>Interchange key.</i> A key used to encrypt a data exchange key (DEK). In Privacy Enhanced Mail, the interchange key is based on public key cryptography.
IMAP2	<i>Interactive Mail Access Protocol.</i> Limited Use Internet protocol defined in RFC 1064.
implicit binding	A method of specifying a binding declaration denoted by the use of a binding external variable declaration rather than a binding parameter declaration. [Netwise RPC Tool]
IMS	<i>Information Management System.</i> Database management software from IBM based on the hierarchical data management model.

index	A direct access method to data. An index has a key value and a pointer to the row of the table that contains data with the key value. An index can be a primary index, where the index is part of the storage structure of the actual table, or a secondary index, which is a separate table in the database with pointers to the base table.
index macro	An expression macro that specifies an index into an array. Index macros begin with the pound sign (#). [Netwise RPC Tool]
Ingres	A popular relational database management system that runs on a variety of operating system platforms. Previously a famous 19th century French painter.
initialization packet	A token ring packet used when a node joins the network.
INOC	<i>Internet Network Operations Center.</i> The management center of some administrative domain in the Internet.
input/output	Generic term for transfer of data from main memory to either a disk drive, terminal, printer, or other device.
instance	The data representing a particular object. As used by the MDB, definitions of instances are called instance records. [SunNet Manager]
instance file	A collection of agent requests and element instance records. [SunNet Manager]
INTAP	<i>Interoperability Technology Association for Information Processing.</i> Japanese Group.
Integrated Services Digital Network	An emerging international communications standard that allows the integration of voice and data on a common transport mechanism.
Intel	Makers of the 80x86 microprocessors used in the IBM PC.
interchange key	A cryptographic key shared by two or more parties.
interface specification	The assignment of values to one or more interface variables using set statements. [Netwise RPC Tool]

<b>interface variable</b>	A variable that controls the operation of the Netwise RPC Compiler. Defined by a set statement in an interface specification. [Netwise RPC Tool]
<b>Interleaf</b>	A desktop publishing program that runs on VAXstations and Sun workstations.
<b>Intermediate System</b>	An OSI system that performs routing and relaying functions in order to provide paths between end systems. Intermediate systems have no functionality above the network layer (although a practical realization of an OSI router will have some amount of end system functionality for network management functions, among other things). Basically equivalent to an Internet Router. [RFC 1136]
<b>International Code Designator</b>	An ISO-administered four-digit unique ID based on organizations (as opposed to the data country code which is based on geographic location).
<b>International Organization for Standardization</b>	International standards-making body, responsible for the Open Systems Interconnection network architecture.
<b>International Telegraph and Telephone Committee</b>	English name for the CCITT.
<b>Internet</b>	A collection of networks that share the same namespace and use the TCP/IP protocols. The Internet consists of at least 400 connected networks. The Internet should not be confused with the internet (lowercase) which refers to all networks that have a path between them. All members of the Internet use TCP/IP.
<b>internet</b>	A collection of networks connected together with gateways and routers. Often refers to the vague mix of heterogeneous networks that are accessible by electronic mail using gateways.
<b>internet address</b>	An assigned number which identifies a host in an internet. It has two or three parts: network number, optional subnet number, and host number. [RFC 1177]



<b>Internet Autonomous System</b>	An autonomous system consists of a set of gateways, each of which can reach any other gateway in the same system using paths via gateways only in that system. The gateways of a system cooperatively maintain a routing database using an interior gateway protocol (IGP). [RFC 1136]
<b>Internet Protocol</b>	The network layer protocol for the Internet. It the datagram protocol defined by RFC 791. [RFC 1177]
<b>internetwork</b>	A collection of data links and the network layer programs for routing among those data links.
<b>internetwork address</b>	An address consisting of a network number and a local address on that network. Used by the network layer for routing packets to their ultimate destinations.
<b>Internetwork Datagram Protocol</b>	Network layer protocol in XNS.
<b>Interpress</b>	Page description language in XNS used for driving laser printers.
<b>interprocess communication</b>	Communication between two processes by passing parameters and return values. Remote calls are a special case of an interprocess communication mechanism.
<b>I/O</b>	<i>See input/output.</i>
<b>IOCTL</b>	<i>I/O Control.</i> Unix function call used to control a device.
<b>IOP</b>	<i>Input/output processor.</i> Term used by Unix-based minicomputer manufacturers to refer to the I/O subsystems.
<b>IP</b>	<i>See Internet Protocol.</i>
<b>IP-ARC</b>	<i>Internet Protocol on ARCnet.</i> Elective Internet protocol defined in RFC 1051.
<b>IPC</b>	<i>See interprocess communication.</i>
<b>IPDS</b>	<i>Intelligent Printer Data Stream.</i> A component of IBM's System Application Architecture(SAA).
<b>IP-FDDI</b>	<i>Transmission of IP over FDDI.</i> Elective Internet protocol defined in RFC 1103.

IP-HC	<i>Internet Protocol on HYPERchannel.</i> Elective Internet protocol defined in RFC 1044.
IPM	<i>Interpersonal Message.</i> The part of the X.400 protocols that defines what the header of a message looks like. The message transfer service (MTS) then defines what the envelope that the message goes in looks like.
IPMS	<i>Interpersonal Messaging System.</i> The protocols used for two user agents to exchange information.
IPN	<i>Interpersonal Notification.</i> An X.400 message type used to exchange information such as receipts.
IP-SLIP	<i>Transmission of IP over Serial Lines.</i> Elective Internet protocol defined in RFC 1055.
IRSG	<i>Internet Research Task Force Steering Group.</i>
IRTF	<i>Internet Research Task Force.</i> The IRTF is a community of network researchers, generally with an Internet focus. The work of the IRTF is governed by its Internet Research Steering Group (IRSG). [RFC 1177]
IS	<i>See intermediate system.</i>
ISAM	<i>Indexed sequential access method.</i> A file structure that allows random access to data via an index and then sequential access to data after that.
ISDN	<i>See Integrated Services Digital Network.</i>
ISH	<i>Intermediate System Hello.</i>
ISH PDU	<i>Intermediate System Hello PDU.</i> From OSI Network Service definition.
ISIS	<i>Intermediate System-Intermediate System.</i>
ISM	<i>Information security manager.</i> The data police.
ISO	<i>See International Organization for Standardization.</i>
ISODE	<i>ISO Development Environment.</i> Software developed by Marshall Rose of PSI International that allows OSI services to use a TCP-based network.

ISPF	<i>Interactive System Productivity Facility.</i> An IBM product that runs on the TSO and CMS user environments. ISPF provides a series of menus (dialogues) for use on a 3270 terminal that allow the user to bypass a command language interface to the operating system.
ISV	<i>Independent Software Vendors.</i> Third-party software developers. Sometimes known as VARs, OEMs, or even “third-party software developers.”
ITSU	<i>Information Technology Standards Unit.</i>
ITT	<i>See Invitation to Transmit.</i>
JANET	<i>Joint Academic NETWORK.</i> JANET is the primary academic network in the United Kingdom, linking about 1000 computers at about 100 universities and research institutes. JANET has a domain name system similar to that of the Internet, but the order of the domain name parts is opposite (with the top-level domain on the left). [RFC 1168]
JCL	<i>Job Control Language.</i> Language used for batch processing on IBM mainframes.
JES	<i>Job Entry Subsystem.</i> Types of processes on IBM mainframes that accept JCL.
JNT	<i>Joint Network Team.</i> The group that maintains the UK academic network.
job queue	A method of letting multiple requests for a scarce resource queue up, as in the case of a print queue.
join	A term used in relational databases to denote two or more tables being combined together.
JSR	<i>Jump-subroutine instruction.</i> An instruction in a program that transfers control to a subroutine.
JTM	<i>Job transfer and manipulation.</i> An OSI layer 7 standard similar in function to a remote job entry (RJE) service.
JvNC	<i>John von Neumann National Supercomputer Center.</i>
Kb	<i>Kilobit.</i> One thousand bits.
KB	<i>Kilobyte.</i> One thousand bytes.

<b>kbps</b>	<i>Kilobits per second.</i> Thousand bits per second.
<b>kbyte</b>	<i>Kilobyte.</i> One thousand bytes.
<b>KDC</b>	<i>Key distribution center.</i> An entity that distributes symmetric keys to two parties. Used in Kerberos.
<b>keep alive message</b>	A message sent over a network link during periods when there is no traffic between users. The message tells the remote node that this computer is still in operation.
<b>Kerberos</b>	A component of MIT's Project Athena. Kerberos is the security system, based on symmetric key cryptography. Contrast with the RSA public key cryptography techniques.
<b>Kermit</b>	A popular file transfer protocol developed by Columbia University. Because Kermit runs in most operating environments, it provides an easy method of file transfer.
<b>kernel</b>	A term used in operating systems. The kernel shields the low-level functioning of the operating system from high-level interfaces, such as user shells.
<b> kerning</b>	A term used in typography which refers to the process of putting variable-size spaces between letters and words to produce an even alignment.
<b>KNET</b>	<i>Kangaroo Network.</i> Hardware/software product made by Spartacus and Fibronics that enables IBM mainframes to communicate over networks with the TCP/IP protocol suite. [RFC 1177]
<b>LAN</b>	<i>Local area network.</i> Usually refers to Ethernet or token ring networks.
<b>LAP</b>	<i>Link access procedure.</i> A procedure for accessing an HDLC data link. Examples are LAPB used in the X.25 environment, and LAPD used in the ISDN environment.
<b>LAPB</b>	<i>Link access procedure, balanced.</i> A subset of the HDLC data link standards.
<b>large nonmoveable cabling system</b>	Another great product name from IBM. <i>See small moveable cabling system.</i>

kbps     large nonmoveable cabling system



LAT	<i>See Local Area Transport.</i>
latency buffer	Token ring concept. The buffer is maintained by the active monitor and is used to compensate for variations in the speed of data on the network.
LAVC	<i>Local Area VAX Cluster.</i> An adaptation of the System Communication Architecture (SCA) to run over the Ethernet instead of a CI bus. Used to enable MicroVAXs to operate as diskless nodes.
LBL	<i>Lawrence Berkeley Laboratory.</i>
LF	<i>Line Feed.</i>
LFN	<i>Long, fat pipe network.</i> A network with high bandwidth and long delay, resulting in a large number of unacknowledged bits. This type of network is known as an "LFN," which is pronounced elephan(t).
LHS	<i>Left Hand Side.</i>
LI	<i>Length Indicator.</i> Used in the ES-IS protocols in the OSI network layer.
Limited Use Protocol	Protocol status for Internet standards. These protocols are for use in limited circumstances. This may be because of their experimental state, specialized nature, limited functionality, or historic state. [RFC 1140]
LISP	<i>List Processing Language.</i>
little endian	A multiple-octet piece of data where the lowest addressed octet is the least significant.
LME	<i>Layer Management Entity.</i>
Local Area Transport	Protocol developed by DEC for communication between terminal servers and hosts on an Ethernet.
LocalTalk	Data link developed for AppleTalk that uses ordinary twisted pair cabling.
lock manager	Part of the operating system that ensures that multiple requests for the same data are not serviced in a way that will damage the integrity of the data.

locking	Preventing another user from accessing a piece of data.
logical	Without reference to physical details. Asking for data in a logical manner, for example, means not having to know where the data is located or how to get it.
Logical Link Control	The upper portion of the data link layer, defined in the IEEE 802.2 standard. The logical link control layer presents a uniform interface to the user of the data link service, usually a network layer. Underneath the LLC sublayer of the data link layer is a media access control sublayer. The MAC sublayer is responsible for taking a packet of data from the LLC and submitting it to the particular data link being used (such as Ethernet or token ring).
LPP	<i>Lightweight Presentation Protocol.</i> A streamlined ASN.1 presentation service defined in RFC 1085 and used in CMOT (RFC 1095). LPP allows the use of ISO presentation services over both TCP and UDP. This approach eliminates the necessity of implementing ISO presentation, session, and transport protocols for the sake of doing ISO network management in a TCP/IP environment. [RFC 1095]
lpr	<i>Line Printer.</i>
LSP	<i>Link state packet.</i> Routing control information message exchanged in a Phase V DECnet or OSI IS-IS routing domain.
LU	<i>Logical unit.</i> An IBM term in SNA which refers to a software or microcode program that uses the network. For example, a terminal connected to a 3274 cluster controller is represented by a LU2 on that cluster controller.
LU 6.2	<i>Logical Unit 6.2.</i> See <i>Advanced Program-to-Program Communication</i> .
LWSP	<i>White Space.</i>
MAC	See <i>Medium Access Control</i> or <i>Macintosh</i> .
Macintosh	A computer made by Apple Computer that is characterized by the graphical, intuitive user interface.
Macintosh Finder	The portion of the Macintosh operating system that manages the desktop.

---

MAC-layer bridge	A device that connects two or more similar data links in a way that is transparent to the user of the data link service (the network layer).
mail-11	The original mail routing protocol used on VMS mail. MAILbus is a more modern message handling architecture.
MAILbridge Server	A series of SoftSwitch products used for connecting together different message handling environments.
MAILbus	A DEC architecture which provides a common message-handling system on a DECnet.
Maintenance Operation Protocol	Special-purpose DECnet protocol used for remote booting on the network and attaching a console onto a station remotely.
managed object	A particular resource (computer system, network interface, etc.) subject to management. In the console, this concept is also called an element. [SunNet Manager]
manager	The collection of programs or processes that work on behalf of a user to help manage a particular set of networks and devices. Usually sends requests to agents, accepts collected data from them, and displays the data for the user. Also called manager application or manager process. [SunNet Manager]
Manager Services	The function library that allows agents and manager to communicate, as viewed by the manager. [SunNet Manager]
manager station	The workstation of a human manager of a particular set of networks and devices, usually running a manager application such as SunNet Manager Console (SNM). [SunNet Manager]
MAP	<i>Manufacturing Automation Protocol</i> . An OSI-related application originally sponsored by General Motors and endorsed by a great many users and vendors.
marshalling	Term used in RPC calls. Taking a local procedure call and packaging it into a form for sending over the network.

MAU	<i>See multistation access unit.</i>
MAXFILELEN	<i>Maximum File Length.</i>
Mb	<i>Megabit.</i> $2^{20}$ bits of information (usually used to express a data transfer rate; as in, 1 megabit/second = 1Mbps). [RFC 1177]
MB	<i>Megabyte.</i> Million bytes of information.
Mbps	<i>Million bits per second.</i>
Mbyte	<i>Megabyte.</i> Million bytes of information.
MC68000	A series of CPU chips manufactured by Motorola. These form the basis for many Unix-based workstations, including the MAC II and the Sun 3 series.
MCB	<i>Message control block.</i> The header portion of a message.
MCC	<i>Management Control Center.</i> <i>See</i> DECMcc.
MCI	Long distance telephone company.
MCI Mail	Commercial electronic messaging service.
MCP	<i>Multiprotocol Communications Processor.</i> Medium-speed communications controller for Sun servers. <i>See also</i> HSI.
MD	A DNS resource record type. Obsolete by MX.
MDB	<i>Management database.</i> The collection of all managed object attribute definitions, along with data definitions specific to the SunNet Manager Console. [SunNet Manager]
Media Interface Connector	The optical fiber connector which connects the fiber to the FDDI controller.
medium	The physical cable, such as coaxial cable, used on a network. Somebody who can speak to the other world.
Medium Access Control	The bottom half of the ISO data link layer. <i>See also</i> Logical Link Control.
MEN	<i>Management event notification.</i> Part of CMIP. Used for sending information about events across the network.



message handling system	A system of protocols, such as X.400, used to exchange messages, such as electronic mail.
Message Integrity Check	A quantity sent along with a message that is derived from the message contents. The MIC is used to verify that the message has not been changed. The MIC has the characteristic of giving very different results when small changes are made to the message contents.
Message Router	DEC product which implements the MAILbus architecture. Message Router is analogous to the X.400 message transfer agent.
message transfer agent	An X.400 term referring to the collections of network members responsible for transferring messages. The final MTA delivers the message to a user agent which is concerned with reading, editing, and other types of interaction with the end user.
MF	A DNS resource record type. Made obsolete by MX.
MHS	<i>See message handling system.</i>
MIB	<i>Management Information Base.</i> A set of definitions of information that a managed object makes available to directors.
MIC	<i>See Message Integrity Check or Media Interface Connector.</i>
MicroVAX	A series of DEC processors usually used as workstations or small servers that compete in the marketplace with Sun and Hewlett-Packard/Apollo.
Milnet	<i>Military Network.</i> A network used for unclassified military production applications. It is part of the Internet. [RFC 1177]
MIN	<i>Minimum.</i>
Minitel	A French terminal used for Videotex applications.
MIPS	<i>Million instructions per second.</i> A measure of the speed of a CPU.

<b>MIR</b>	<i>Management information repository.</i>
<b>mirrored</b>	A disk drive is mirrored when two identical copies of the data are kept on two different disk drives. If one fails, the other can keep on operating.
<b>MIS</b>	<i>Management Information System.</i> A database system used to provide information to managers in an organization. The term has come to refer to the department in an organization responsible for computing.
<b>MIT</b>	<i>Massachusetts Institute of Technology.</i> Developers of the X Window System. Also a university.
<b>mixed buffering style</b>	A buffering style in which the argument or result PDU is packed in a single, contiguous, memory-resident buffer, and the request or reply PDU is packed according to the packet buffering style. [Netwise RPC Tool]
<b>MLAL</b>	<i>Multiletter acronym listing.</i>
<b>MMDF</b>	<i>Multichannel Memo Distribution Facility.</i> A mail handler on Unix systems.
<b>mode</b>	<i>See external variable mode.</i>
<b>modem</b>	<i>Modulator/demodulator.</i> A device that takes digital data from a computer and encodes it in analog form for transmission over a phone line. Modems are also used to connect computers to an analog broadband system.
<b>monitor bit</b>	Token ring concept. The monitor bit is flipped by the active monitor to prevent a frame of priority greater than 0 from circulating continuously.
<b>MOP</b>	<i>See Maintenance Operation Protocol.</i>
<b>MOTIS</b>	<i>Message Oriented Text Interchange System.</i> Formal name for the 1988 CCITT X.400 standards.
<b>mount</b>	The process of making a file system available to the local file system. A mount system call is used to inform the kernel about a new file system. If the file system is remote, the NFS mount protocol is used.
<b>mouse</b>	A pointing device used on workstations. A small rodent.

---

MPR	<i>Multiport repeater.</i>
MPT	<i>Multiport transceiver.</i>
MS	<i>Mail Stop.</i>
MT	<i>Message Transfer.</i>
MTA	<i>See message transfer agent.</i>
MTS	<i>Message transfer service.</i> The X.400 protocols that govern the exchange of envelopes of information. The IPMS defines the content of the envelope.
MTTF	<i>Mean Time to Failure.</i> The average time between hardware breakdown or loss of service. This may be an empirical measurement or a calculation based on the MTTF of component parts. [RFC 1177]
MTTR	<i>Mean Time to Recovery.</i> The average time it takes to restore service after a breakdown or loss. This is usually an empirical measurement. [RFC 1177]
MTU	<i>Maximum transfer unit.</i> The biggest piece of data that can be transferred on a given subnetwork.
multicast	An address to which several nodes will respond. Contrast to broadcast, where all nodes on a network will respond.
multicasting	A term used in Ethernet addressing. A multicast address is a group address that is meant for a certain subset of users on the Ethernet. LAT nodes communicate their current status with each other using a multicast address. To be contrasted with a broadcast address which is received by all users on the Ethernet.
multilink end node	An end node which has more than one connection to the network. The end node may send and receive data over any of the links but will not route traffic for other nodes.
multiplexing server	A method of handling binding requests in which a single server application process itself is responsible for all RPC requests over all bindings. [Netwise RPC Tool]

<b>multipoint</b>	A data link layer concept in which multiple nodes share a common physical medium. In a multipoint situation, a single node is the controller of the line and polls all tributaries periodically to see if they wish to send data. This is in contrast to multiaccess media like Ethernet where any node may send without permission.
<b>multiport repeater</b>	An Ethernet repeater, typically for ThinWire networks, that connects several segments together into a multisegment Ethernet.
<b>multiport transceiver</b>	Several Ethernet transceivers built into one device. Can operate as a concentrator on a cable or as a stand-alone Ethernet (known as Ethernet in a Can).
<b>multiprocessing server</b>	A method of handling binding requests in which the server application process spawns a new process responsible for all RPC requests over a given binding. [Netwise RPC Tool]
<b>multisegment Ethernet</b>	Several segments of Ethernet connected together with repeaters. All signals broadcast on a multisegment Ethernet are received by all other nodes. This is in contrast to the extended Ethernet, where the MAC-layer bridge forwards only those packets destined for the other Ethernet.
<b>multistatement transaction</b>	Several different interactions with the database that are grouped into a single transaction. If any one of the operations is not carried out because of a user abort or system crash, the entire transaction is rolled back. In a multistatement transaction all or none of the operations is carried out.
<b>multistation access unit</b>	Token ring device used to connect several stations to the ring. Similar to the multiport transceiver for the Ethernet.
<b>multithreaded</b>	An operating system feature that allows a process to maintain several threads of execution, each under the control of the parent process. OS/2 is an example.
<b>multithreading server</b>	A method of handling binding request in which a single server application process creates a new thread responsible for all RPC requests over a given binding. [Netwise RPC Tool]
<b>MVS/TSO</b>	<i>Multiple virtual storage/time sharing option.</i> MVS is an IBM operating system. TSO is the interactive subsystem, as opposed to a system like JES that is used for batch processing.



<b>MX</b>	<i>Mail Exchange.</i> A DNS resource record type indicating which host can handle mail for a particular domain.
<b>NAK</b>	<i>Negative acknowledgment.</i> Response to nonreceipt or receipt of a corrupt packet of information.
<b>Name Binding Protocol</b>	AppleTalk protocol for mapping logical names to network addresses.
<b>Named Pipes</b>	A process-to-process protocol that allows a full-duplex communication path to be maintained. The pipe is the end-point of the communication path, through which a process gains entry to the function. Names are maintained and registered on the network, allowing a pipe to access services.
<b>namedroppers</b>	A mailing list concerned with the Domain Name System.
<b>NAMESERVER</b>	<i>Host Name Server Protocol.</i> Obsolete Internet protocol defined in IEN-116.
<b>namespace</b>	A commonly distributed set of names in which all names are unique.
<b>Namespace Creation Timestamp</b>	A unique ID for a namespace which distinguishes it from all other namespaces.
<b>NAS</b>	<i>See Network Application Support.</i>
<b>NASA</b>	<i>National Aeronautics and Space Administration</i>
<b>National Bureau of Standards</b>	<i>See National Institute for Standards and Technology.</i>
<b>National Institute for Standards and Technology</b>	United States governmental body that provides assistance for standards-making. Formerly the National Bureau of Standards.
<b>NAU</b>	<i>Network addressable unit.</i> The boundary of an IBM SNA network. Logical and physical units are examples of NAUs.
<b>NBP</b>	<i>See Name Binding Protocol.</i>
<b>NBS</b>	<i>National Bureau of Standards. See National Institute for Standards and Technology.</i>

NBS-AS2	<i>National Bureau of Standards, Abstract Syntax 2.</i> An abstract syntax developed by the NBS (now NIST) to make a file directory an entry that appears like a file, allowing remote FTAM systems to request a directory of files.
NCS	<i>See Network Computing System.</i>
ND	<i>NetDisk.</i> A Sun protocol for loading a raw disk over the network, similar in function to DEC's MOP protocols. Now obsolete.
NEARnet	<i>New England Academic and Research Network.</i>
NetBIOS	<i>Network Adapter Basic Input/Output System.</i> A network protocol that allows a client program to find a server process and communicate with it. Similar to Named Pipes.
NETBLT	<i>Bulk Data Transfer Protocol.</i> Obsolete Internet high-speed block transfer protocol defined in RFC 998.
NETSTAT	<i>Network Statistics.</i> A network monitoring protocol defined in RFC 869.
NetWare	The networking components sold by Novell. A collection of data link drivers, a transport protocol stack, workstation software, and the NetWare operating system.
NetWare Core Protocols	Protocols used to obtain the core services offered by a NetWare file server. Includes a variety of facilities such as file access, locking, printing, and job management.
Netwise	Maker of the Netwise RPC Tool.
Network	A single Internet network (which may or may not be divided into subnets). [RFC 950]
network address	The number of the network that the user is on. Each network (data link) in an internetwork has a number assigned to it. The full address of a station is the network address plus the local address of the node on that network.
Network Application Support	DEC standards to allow everything to work with everything. Similar in scope and effectiveness to IBM's SAA.
Network Computing System	Apollo's computing architecture. The DEC RPC mechanism is derived from the NCS RPC architecture.

Network File System	A distributed file system developed by Sun Microsystems and widely used on TCP/IP systems.
Network Information Service	Name service in the Sun Open Network Computing (ONC) family. <i>See also Yellow Pages.</i>
network library	Software module that comes with the Netwise RPC Tool. The network library is added to the code generated by the RPC Compiler to form a complete program. The library masks access to network communications from the rest of the program.
Network Loadable Module	Program on NetWare 386 that when loaded becomes a part of the operating system.
network number	The part of an internet address which designates the network to which the addressed node belongs. [RFC 1177]
network selector	The part of an address at layer N which selects a particular user at layer N. In other words, the address of the next layer.
Network Service Access Point	A conceptual point on the Network/Transport Layer boundary in an end system that is globally addressable (and the address globally unambiguous) in OSI. An NSAP represents a service available above the Network Layer (such as a choice of transport protocols). An End System may have multiple NSAPs. An NSAP address is roughly equivalent to the Internet [address, protocol] pair. [RFC 1136]
Network Services Protocol	DECnet transport layer protocol.
NeWS	<i>Network Extensible Window System.</i> A windowing environment from Sun Microsystems based on the PostScript language and a proprietary window control protocol.
NEXT	Computer company founded by Steve Jobs, formerly of Apple.
NFBLK	<i>NFS Block.</i> A file attribute that indicates that the file is a block special file.
NFCHR	<i>NFS Character.</i> A file attribute that indicates that the file is a character special file.

NFDIR	<i>NFS Directory.</i> A file attribute that indicates that this is a directory.
NFLNK	<i>NFS Link.</i> A file attribute that indicates that this is a symbolic link.
NFNON	<i>NFS Named Socket.</i> A file attribute that indicates that this is a named socket.
NFREG	<i>NFS Regular File.</i> A file attribute that indicates that this is a regular file.
NFS	<i>Network File System.</i> A network service that lets a program running on one computer use data stored on a different computer on the same internet as if it were on its own disk. [RFC 1177]
nibble	Half a byte.
NIC	<i>Network Information Center.</i> A facility located at the Stanford Research Institute that administers Internet addresses.
NICE	<i>Network Information and Control Exchange.</i> A DECnet protocol used for the exchange of network management information.
NIS	<i>Network Information Services.</i> See Yellow Pages.
NIST	<i>See National Institute for Standards and Technology.</i>
NLM	<i>See Network Loadable Module.</i>
NNSC	<i>NSF Network Service Center.</i>
NNTF	<i>Networking and Telecommunications Task Force.</i>
NOC	<i>Network Operations Center.</i> An organization which is responsible for maintaining a network. [RFC 1177]
node	An individual item in a set. An Ethernet node, for example, is a device attached to the cable with a transceiver, including a repeater, bridge, or computer. A file system node is a directory or individual file.
nonexclusive lock	A type of lock on a file that permits other users to read information but prevents any write operations.



nonpersistent binding	A style of binding in remote procedure calls where the connection is set up and torn down every time the remote procedure is called.
nonpersistent style	A method of controlling binding. When this style used by a remote procedure declaration, the Netwise RPC Compiler generates code to support a nonpersistent binding for that remote procedure. [Netwise RPC Tool]
nonrestricted token	FDDI token in asynchronous mode available to all users.
nonrouting node	<i>See end node.</i>
NOOP	<i>No Operation.</i>
no-reply calling style	A calling style in which the server stub state machine does not generate a reply and the client stub state machine does not wait for a reply. [Netwise RPC Tool]
Not Recommended Protocol	Protocol status for Internet standards. These protocols are not recommended for general use. This may be because of their limited functionality, specialized nature, or experimental or historic state. [RFC 1140]
Novell	Makers of NetWare software for networks.
NPDU	<i>Network Protocol Data Unit.</i>
NPID	<i>Network Layer Protocol Identifier.</i> Used in the ES-IS protocols in the OSI network layer.
NPSI	<i>Network Packet Switching Interface.</i> A type of IBM software that allows SNA data to be carried over an X.25 network to another SNA environment.
N(R)	<i>Receive sequence number.</i> LLC field that indicates the sequence number of the last packet received.
NREN	<i>National Research and Education Network.</i>
NRS	<i>Name Registration Scheme or Next Receive Sequence.</i> The Name Registration Scheme is the U.K. equivalent to the Domain Name System. The Next Receive Sequence is a VMTP flag indicating that the associated request message (in a response) or previous response (if a request) was received consecutively with the last request from this entity. That is, there were no interfering messages received.

N(S)	<i>Send sequence number. LLC field that indicates the sequence number of the packet being sent.</i>
NSAP	<i>See Network Service Access Point.</i>
NSDU	<i>Network Service Data Unit.</i>
NSF	<i>National Science Foundation</i>
NSFnet	<i>National Science Foundation Network. A high-speed internet that spans the country, and is intended for research applications. It is made up of the NSFnet Backbone and the NSFnet regional networks. It is part of the Internet. [RFC 1177]</i>
NSFnet Backbone	<i>A network connecting 13 sites across the continental United States. It is the central component of NSFnet. [RFC 1177]</i>
NSFnet Regional	<i>A network connected to the NSFnet Backbone that covers a region of the United States. It is to the regionals that local sites connect. [RFC 1177]</i>
NSI	<i>NASA Science Internet. NASA's wide-area network.</i>
NSP	<i>See Network Services Protocol.</i>
NSS	<i>Main routing nodes in the NSFnet backbone.</i>
NVP	<i>Network Voice Protocol. Limited Use Internet protocol defined in an ISI-memo.</i>
NVT	<i>Network Virtual Terminal. The virtual terminal defined in the Telnet service.</i>
NWNET	<i>NorthWestNet.</i>
NYSERnet	<i>New York State Educational and Research Network. An internet which serves New York educational and research institutions. It also serves as the NSFnet regional network for New York State. [RFC 1177]</i>
OA	<i>Office automation. Whatever's in Datamation this month.</i>

---

OCR	<i>Optical Character Recognition.</i>
OEM	<i>Original equipment manufacturer.</i> Company that sells equipment which is embedded in another company's products. The other company is known as a value-added reseller (VAR) of the product.
OID	<i>Object ID.</i>
OIW	<i>OSI Implementors Workshop.</i> A series of workshops hosted by the NIST.
OLTP	<i>On-line Transaction Processing.</i> A term muddled by marketing departments to mean all that is good and fast. Really means any application that executes lots of small transactions rapidly.
ONC	<i>See Open Network Computing.</i>
opaque	A data type not interpreted by a program. Often used to store credentials for security applications or a file handle for NFS.
Open Network Computing	Sun marketing term for the family of protocols that include the Network File System.
Open Software Foundation	Nonprofit organization founded by Digital, IBM, and four other vendors to develop specifications for an open software environment.
Open Systems Interconnection	The ISO standards for a heterogeneous, open network architecture.
Open Systems Interconnection Environment	The global collection of Open Systems. Basically equivalent to the Internet. [RFC 1136]
O/R address	<i>Originator/recipient address.</i> A valid X.400 address.
OS	<i>Operating system.</i>
OS/2	<i>Operating System/2.</i> Successor to DOS for the IBM PC (assuming anybody wants it, that is).

OSF	<i>See Open Software Foundation.</i>
OSI	<i>See Open Systems Interconnection.</i>
OSIE	<i>See Open Systems Interconnection Environment.</i>
OSI Reference Model	An "outline" of OSI which defines its seven layers and their functions. Sometimes used to help describe other networks. [RFC 1177]
OSPF	<i>Open Shortest-Path First.</i> An experimental replacement for RIP. It addresses some problems of RIP and is based on principles that have been well-tested in non-Internet protocols. [RFC 1177]
OU	<i>Organization Unit.</i> An X.400 address attribute indicating a subunit of an organization.
outgoing connection timer	Session control term. When the session layer issues a connect request to the transport layer, it starts this timer. If an accept or reject indication is not received before the timer expires, session control issues a disconnect and informs the end user.
Pack	Term used in remote procedure calls for translating data from the machine-dependent format into a machine-independent format.
pack procedure	A procedure created by the Netwise RPC Compiler that encodes one or more data types into a PDU in a standard format for transmission across a network. [Netwise RPC Tool]
packet	The unit of data sent across a packet switching network. The term is used loosely. While some Internet literature uses it to refer specifically to data sent across a physical network, others view the Internet as a packet switching network and describes IP datagrams as packets. [RFC 1177]
Packet Assembler/Disassembler	Special-purpose computer on an X.25 network that allows asynchronous terminals to use the synchronous X.25 network by packaging asynchronous traffic into a packet.
packet buffering style	A buffering style in which an entire PDU usually does not exist in memory at any one time. Individual portions of the PDU are successively packed and unpacked in a relatively small fixed-size buffer. This style allows PDU transmission to occur in parallel with the packing and unpacking. [Netwise RPC Tool]



<b>Packet Exchange Protocol</b>	An XNS transport protocol that requires each packet to be separately acknowledged. A PEP-like protocol forms the foundation of the NetWare Core Protocols.
<b>packet switching</b>	A network that has packaged data into packets. A computer can handle many more virtual connections with packets than it can with dedicated connections (known as circuit switching). Packet switching forms the basis for X.25, as well as most network-layer protocols.
<b>PAD</b>	<i>See Packet Assembler/Disassembler.</i>
<b>paging</b>	A memory management technique in a virtual memory operating system. Only a few parts (pages) of a program are actually in memory. When a new part is needed, it is paged into memory.
<b>PAP</b>	<i>See Printer Access Protocol.</i>
<b>PAR</b>	<i>Positive acknowledgment retransmit.</i> A method of assuring reliable communications used by the DDCMP data link protocol.
<b>Paradox</b>	Popular database program for the IBM PC.
<b>PARC</b>	<i>See Xerox Palo Alto Research Center.</i>
<b>path</b>	As a file system concept, the path indicates what set of folders or subdirectories a file is stored in. In the networking sense, a path is the route that a packet takes from the source to the destination. The path is a series of data links or hops.
<b>PBR</b>	<i>Policy-based routing.</i>
<b>PBX</b>	<i>Private branch exchange.</i> A telephone switch which is installed at the customer premises.
<b>PC</b>	<i>Personal computer.</i> A desktop computer developed by IBM or a clone developed by a third-party vendor. PC is sometimes used more generically to refer to other desktop systems, such as the Apple Macintosh.
<b>PC-DOS</b>	<i>Personal Computer-Digital Operating System.</i> IBM's version of Microsoft's operating system.

PC-NFS	<i>Personal Computer Network File System.</i>
PD	<i>Protocol Draft.</i>
PDL	<i>Page description language.</i> PostScript is an example of a PDL.
PDP	<i>Programmable data processor.</i> A series of Q-bus-based 16-bit minicomputers manufactured by DEC.
PDS	<i>Postal Delivery System or Premises Distribution System.</i> The Postal Delivery System is also known as snail mail. Premises Distribution System is an AT&T cabling system.
PDU	<i>See protocol data unit.</i>
PEP	<i>See Packet Exchange Protocol.</i>
permanent virtual circuit	A circuit that is kept up permanently, as in the case of a dedicated leased line on the telephone network.
persistent binding	A binding established between application processes on a network that is designated to be used by more than one remote procedure call. [Netwise RPC Tool]
persistent close style	A method of controlling binding. When this style is used by a remote procedure declaration, the Netwise RPC Compiler generates code to use an existing persistent binding and to terminate it upon return of the remote procedure. [Netwise RPC Tool]
persistent open style	A method of controlling binding. When this style is used by a remote procedure declaration, the Netwise RPC Compiler generates code to establish a persistent binding. [Netwise RPC Tool]
persistent use style	A method of controlling binding. When this style is used by a remote procedure declaration, the Netwise RPC Compiler generates code to use an existing persistent binding. [Netwise RPC Tool]
PEX	<i>See PHIGS Extension to X.</i>
PHIGS	<i>See Programmer's Hierarchical Interactive Graphics System.</i>
PHIGS Extension to X	Extension to the X Window System to incorporate 3D graphics.

PHY	<i>Physical.</i>
PICS	<i>Protocol Implementation Conformance Statement.</i>
PID	<i>Process identification number or protocol ID.</i> The process identification number is used to identify each process running on an operating system. The protocol ID is used to indicate the next-level-up user of a service.
piggybacked	Added on to. A term used in protocols that require the acknowledgment of prior packets. The acknowledgment can often be piggybacked into the same packet as data that is headed in that direction.
ping-pong	A type of transport protocol that requires each packet to be individually acknowledged. Before a node can send another packet, it must wait for an acknowledgment. PEP is an example of this, as opposed to SPX, which allows a window of unacknowledged packets to be outstanding.
pipe	<i>See Named Pipes.</i>
PN	<i>Personal Name.</i> An X.400 address attribute indicating a person's name.
pointer macro	An expression macro that specifies a pointer to either an element of a variable, a structure variable, or a member of a structure variable. Pointer macros begin with a dollar sign. [Netwise RPC Tool]
Pointer Table	A table created by the Netwise RPC code that contains the pointers to unique structures to be encoded in a PDU. This table is used by the RPC code to avoid duplication of structures and allow the passing of circular or self-referential structures to remote procedures. [Netwise RPC Tool]
POP	<i>Post Office Protocol.</i> Messaging protocol described in RFC 1081.
portal	There may be several different users of the Ethernet service, each with a portal, or identification number. Incoming packets are then distributed to the appropriate portal.
POSI	<i>Promotion of OSI.</i> Japanese group.

POSIX	<i>Portable Operating System Interface for Computer Environments.</i> IEEE-developed standards to provide a common interface to the operating system, thus making applications more portable.
PostScript	A page description language used on printers such as the Apple LaserWriter and on computer displays used in workstations from companies such as NeXT and Sun Microsystems. Similar in function to Xerox's Interpress.
POTS	<i>Plain Old Telephone Service.</i> As opposed to ISDN, Call Waiting, or any other modern marvels.
PPDU	<i>Presentation Protocol Data Unit.</i>
ppm	<i>Pages per minute.</i> Rating measure for laser printers.
PRDMD	<i>Private Directory Management Domain.</i> An X.500 administrative entity which is not a public service provider.
Presentation Manager	The component of the OS/2 operating system that manages the user interface.
presentation syntax	A standard method of representing data in a heterogeneous environment. The Abstract Syntax Notation 1 (ASN.1) is an example of a presentation syntax.
primitive	ISO definition: an element of a service provided by one entity to another one.
print spooler	A software program that accepts several jobs at once for printing. The spooler controls access to the printer and queues incoming jobs for execution.
Printer Access Protocol	AppleTalk protocol for accessing printers.
PRMD	<i>Private management domain.</i> An X.400 domain. <i>See</i> ADMD.
PROFS	<i>Professional Office System.</i> IBM office automation package for the VM/CMS operating system.
Programmer's Hierarchical Interactive Graphics System	Imaging system providing sophisticated capabilities such as hidden-surface removal, shading, and depth cueing.



propagation velocity	The rate at which a signal propagates on a wire. Signals travel over a wire much like ripples in a pond after a stone is thrown in.
Proposed Standard Protocol	Classification for Internet standards. These are protocol proposals that may be considered by the IAB for standardization in the future. Implementation and testing by several groups is desirable. Revision of the protocol specification is likely. [RFC 1140]
protocol	A formal description of message formats and the rules two computers must follow to exchange those messages. Protocols can describe low-level details of machine-to-machine interfaces (e.g., the order in which bits and bytes are sent across a wire) or high-level exchanges between allocation programs (e.g., the way in which two programs transfer a file across the Internet). [RFC 1177]
protocol data unit	A layer communicates with its peer by sending packets. Each packet has a header that contains information that the peer will work with, such as addresses or acknowledgment requests. It also contains data, the protocol data unit, that is passed up to the client of the layer.
protocol stack	A set of functions, one at each layer of the protocol stack, that work together to form a set of network services. Each layer of the protocol stack uses the services of the module beneath it and builds on that service.
proxy agent	An agent that manages on behalf of the manager, used to manage objects not directly manageable due to different protocol suites or other incompatibilities. Sometimes shortened to proxy. [SunNet Manager]
proxy system	The host where a proxy agent runs. [SunNet Manager]
PS	<i>Presentation Service.</i>
PS profiles	<i>Presentation services profiles.</i> An SNA term used to allow two NAUs to negotiate an acceptable subset of presentation functions they can both support.

PSC	<i>Pittsburgh Supercomputing Center</i>
PSI	<i>Packet-switch interface or Performance Systems International.</i> Packet-switch Interface is DEC software to allow a VAX to participate in an X.25 network. Performance Systems International is a network service provider and an active participant in the areas of ISODE, X.500, and SNMP.
PSN	<i>Packet-switched network.</i> An X.25 network.
PSTN	<i>Public Switched Telephone Network.</i>
PTR	<i>Pointer.</i> A DNS resource record type used to provide an alias mechanism between one name and another. Often used to point between the address (stored as an IN-ADDR.ARPA record) and the host name.
PTT	<i>Poste Téléphone et Télégraphe.</i> A government provider of communications functions in most European countries.
PU	<i>Physical unit.</i> An SNA term used to refer to different types of hardware in the network. A 3274 cluster controller is a PU type 2 (PU2).
public domain	Intellectual property available to people without paying a fee. Most computer software developed at universities is in the public domain.
PVC	<i>Polyvinyl chloride or see permanent virtual circuit.</i> Polyvinyl chloride is a type of cable coating unsuitable for environmental airspaces but often used in offices.
PVP	<i>Packet video protocol.</i>
Q.700	Introduction to CCITT SS No. 7.
Q.701	The message transfer part of Signalling System No. 7.
Q.711	Signalling connection control part of Signalling System No. 7.
Q.721	Telephone user part of Signalling System No. 7.
Q.730	ISDN supplementary services definition for Signalling System No. 7.
Q.761	The ISDN user part of Signalling System No. 7.

---

Q.930	CCITT standard for the ISDN user-network interface at layer 3.
Q-bus	The peripheral bus used on MicroVAX and PDP computers.
QOS	<i>Quality of service.</i> A series of negotiable parameters in X.25 and OSI network implementations.
Quarterdeck Office Systems	Makers of Desqview/386, a program that allows the 80386 computer running DOS to perform several different tasks.
QUOTE	<i>Quote of the Day Protocol.</i> Elective Internet protocol defined in RFC 865.
R	<i>Reply.</i> Sniffer Analyzer label for a packet that is a reply. C stands for command.
RAM	<i>Random access memory.</i> Dynamic memory, sometimes known as main memory or core.
RARP	<i>Reverse Address Resolution Protocol.</i> A TCP/IP protocol which provides the reverse function of ARP. Used by diskless nodes when they first initialize to find their Internet address.
RCL	<i>Revoked Certificate List.</i> A list used in X.509 security to specify which certificates are now longer valid.
rcp	<i>Remote copy program.</i> An upper-layer TCP/IP service found in the Berkeley Unix implementation for copying files. <i>See FTP.</i>
RD PDU	<i>Redirect PDU.</i> From OSI Network Service Definition. Used to inform a system to redirect future PDUs to another intermediate system. A primitive form of routing control.
RDA	<i>Remote Data Access.</i> An international standard for access to databases in a heterogeneous computing environment.
RDATA	<i>Resource Data.</i> A variable portion of a DNS resource record.
RDN	<i>Routing Domain Number.</i> A portion of an OSI address assigned by the Internet Assigned Numbers Authority.
RDP	<i>Reliable Data Protocol.</i> Limited Use Internet protocol defined in RFCs 908 and 1151.

READDIR	<i>Read Directory.</i> An NFS operation to read the contents of a directory.
READLINK	An NFS operation to find out where a symbolic link points to.
Recommended Protocol	Protocol status for Internet standards. Systems should implement these protocols, but it is not required.
reference	A special type of pointer for which the referenced memory has already been allocated. [Netwise RPC Tool]
REJ	<i>Reject.</i>
reject PDU	A PDU, created by the server RPC code and returned to the client application process, that contains information describing errors detected by the RPC code. [Netwise RPC Tool]
remote procedure	A user-written procedure within a server application process that is called by a client application process. [Netwise RPC Tool]
remote procedure call	A set of network protocols that allow a node to call procedures that are executing on a remote machine. The Netwise RPC Tool, HP/Apollo RPC, and Sun's NFS RPC are examples of such a protocol.
rendezvous	Process where an agent sends data and event reports. Also known as rendezvous process. [SunNet Manager]
repeater	An Ethernet device used to connect two or more segments of cable together. The repeater retimes and reamplifies the signal received on one segment before resending it on all other segments.
report	Answer (usually from an agent to a manager) containing attribute values or error messages. Sometimes called a response. [SunNet Manager]
request	Network management call from a manager to an agent process, usually to collect and return specific attribute values. [SunNet Manager]
request bitmap	An AppleTalk Transaction Protocol field used to keep track of which pieces of a reply have been sent or received.



Request for Comment	The Internet's Request for Comments documents series. The RFCs are working notes of the Internet research and development community. A document in this series may be on essentially any topic related to computer communication and may be anything from a meeting report to the specification of a standard. [RFC 1177]
Required Protocol	Protocol status for Internet standards. A system must implement the required protocols. [RFC 1140]
reservation field	A field in token ring packets that allows a node to inform the active monitor that it has data of a certain priority to send.
resource fork	The second half of a file on a Macintosh file system. The resource fork contains the executable code while the data fork has user data.
restricted token	A special mode of asynchronous access in FDDI where the bandwidth is dedicated to an extended dialogue between two users.
RFC	<i>See Request for Comment.</i>
RG-62	Grade of coaxial cable.
RGB	<i>Red, green, blue.</i> A method of representing colors as a mix of the three primary colors.
RHS	<i>Right Hand Side.</i>
ring purge	A token ring packet that clears the network of data, similar in function to the ARCnet reconfiguration burst.
RIP	<i>See Routing Information Protocol.</i>
RISC	<i>Reduced instruction set computer.</i> Generic name for CPUs that use a simpler instruction set than more traditional designs. Examples are the IBM PC/RT, Pyramid minicomputers, and Sun SPARC workstations.
RJ	<i>Reject.</i> A protocol data unit type in the OSI transport service.
RJ11	Standard modular jack developed by AT&T. Used for telephones and data communications. Being replaced by the RJ45, which is the same size but has more wires.

RJE	<i>Remote job entry.</i> Facility for submitting a job to a computer for execution. Card readers were early RJE stations. Usually means software that emulates RJE stations.
rlogin	<i>Remote log-in.</i> Berkeley TCP/IP command to log onto a remote node.
RLP	<i>Resource Location Protocol.</i> Elective Internet protocol defined in RFC 887.
RM	<i>Record marking.</i> A technique used in the RPC protocol to differentiate one record from another over the streaming TCP protocol.
RMS	<i>Record management services.</i> A common I/O interface for VMS used for access to local data via QIO calls and remote data via the DAP protocol.
root	Unix superuser. The one account on a Unix system that has privileged access.
root directory	The top of a namespace or file system.
rotary	An example of a rotary is when several outgoing lines are available for a dial-out service. Rather than make the user ask for each line by name, a rotary service connects the user to the first available line. A rotary is thus multiple instances of an all-accessible service using one address. The service provider intercepts all requests for that address and farms them out to a specific address.
Router	Dedicated hardware used to route traffic on a network. The alternative is to use a portion of a general purpose system such as a Sun or VAX.
routing directory	A database maintained by the network layer to determine which paths to use to get to particular networks.
Routing Domain	A set of End Systems and Intermediate Systems which operate according to the same routing procedures and which is wholly contained within a single administrative domain. [RFC 1136]
Routing Information Protocol	One protocol which may be used on internets simply to pass routing information between gateways. It is used on many LANs and on some of the NSFnet regional networks. [RFC 1177]

routing table	A directory maintained by the network layer that contains the address of nodes on the internetwork and how to reach them.
Routing Table Maintenance Protocol	AppleTalk protocol for the maintenance of routing tables.
RPC	<i>See remote procedure call.</i>
RPC binding	A logical linkage between client and server application processes for the purpose of invoking remote procedures within a single RPC interface. [Netwise RPC Tool]
RPC Compiler	A source code generator supplied as part of the Netwise RPC Tool. [Netwise RPC Tool]
RPC macro	One of several special macros that the programmer can use to customize the source code generated by the Netwise RPC Compiler. [Netwise RPC Tool]
RPC Server Control Library	A component of the Netwise RPC Tool that consists of a set of routines that manage the RPC aspects of the server application process. [Netwise RPC Tool]
RPC Specification	A description of the interaction between client and server application processes in terms of their use of remote procedure calls. This specification is the input to the Netwise RPC Compiler. [Netwise RPC Tool]
RPC Tool	The RPC mechanism sold by Netwise, including a RPC compiler.
RR	<i>Receive ready or Resource Record.</i> Receive ready is a LLC field indicating that the sending node is ready to receive data. Resource Record is information stored in the Domain Name System. Different types of resource records are available and the system is extensible.
RRQ	<i>Read Request.</i> Packet type in the Trivial File Transfer Protocol.
RS-232-C	A physical interface standard, used frequently for connecting asynchronous devices such as terminals. Developed by the Electronic Industries Association to define the electrical and mechanical link between a DTE and a DCE.

RSA	<i>Rivest, Shamir, and Adleman.</i> Developers of a patented public key cryptography method that forms the basis for the Internet Privacy Enhanced Mail as well as X.509 directory security.
RSADSI	<i>RSA Data Security International.</i> The firm founded by RSA to commercialize their research.
RST	<i>Reset.</i> A flag in the TCP header.
RSTS	<i>Resource-sharing timesharing system.</i> PDP-based operating system.
RSX	Yet another PDP-based operating system. VMS systems running in compatibility mode are able to execute RSX-executable images.
RTL	<i>Run-Time Library.</i>
RTMP	<i>See Routing Table Maintenance Protocol.</i>
RT/PC	IBM 32-bit workstation based on a RISC architecture.
RTS	<i>Reliable Transfer Service.</i>
RTT	<i>Round-trip transmission.</i> A measure of the current delay on a network.
RU	<i>Request unit.</i> A part of IBM's SNA architecture. A series of request units are sent from one session participant to the other; they are then processed by the upper layers of the protocol stack.
RUIP	<i>Remote user information program.</i> A program that can be accessed using the finger protocol defined in RFC 1194.
SAA	<i>Systems Application Architecture.</i> IBM architecture to present common user, communications, and programming interfaces across multiple hardware platforms and operating systems.
SABME	<i>Set Asynchronous Balanced Mode Extended.</i> Type of frame used in the LLC operation.



---

SACK	<i>See selective acknowledgment.</i>
SAP	<i>See Service Access Point.</i>
SAS	<i>See Single Attachment Station.</i>
SASE	<i>Specific application service element.</i> Application layer concept in the OSI network architecture. Refers to special-purpose services such as the job transfer and manipulation (JTM) facility.
Satellite Equipment Room	Consists of a room with patch panels for high- and low-speed data and voice communications as well as active devices such as terminal servers, repeaters, or multiport transceivers.
SCA	<i>System Communication Architecture.</i> The DEC architecture for VAX Clusters.
SCALE	The TCP window scaling option defined in RFC 1072. Allows window information to be interpreted as being scaled by 1 to 16 powers of 2, thus increasing the size of the effective window.
SCCP	<i>Signalling connection control part.</i> <i>See Q.711.</i>
schema	A description of object structures and instances. [SunNet Manager]
schema file	A Unix file containing one or more agent schema. Also called agent schema file. [SunNet Manager]
SCP	<i>See server control procedure or Session Control Protocol.</i>
SCS	<i>System communication services.</i> Software services used in a VAX Cluster to provide internode communication. SCS is the lowest level of the System Communication Architecture.
SCSI	<i>Small computer standard interface.</i> Pronounced "scuzzy." A standard for connecting disk drives to disk controllers, used typically in small multiuser computers.
SDLC	<i>Synchronous data link control.</i> IBM's data link protocol used in SNA networks.

SDU	<i>See Service Data Unit.</i>
search path	A mechanism in DOS, Unix, and other operating systems that allows a user to specify a command without knowing which directory it is stored in. The operating system will search each of the directories in the search path for the command until it finds the file.
selective acknowledgment	A TCP option which is used to convey extended acknowledgment information over an established connection. Specifically, it is to be sent by a data receiver to inform the data transmitter of noncontiguous blocks of data that have been received and queued. The data receiver is awaiting the receipt of data in later retransmissions to fill the gaps in sequence space between these blocks. At that time, the data receiver will acknowledge the data, normally by advancing the left window edge in the Acknowledgment Number field of the TCP header. [RFC 1072]
semaphore	A synchronization mechanism on an operating system.
separable calling style	A calling style in which the client stub state machine separates the processing of remote procedure requests from the processing of replies. This is a prerequisite for asynchronous processing. [Netwise RPC Tool]
sequence number	A unique number for every packet on a particular connection maintained by a reliable transport layer service. The sequence number allows the transport layer to see if any packets were lost or delivered out of sequence by the underlying network and data layers.
Sequenced Packet Protocol	XNS protocol for reliable transfer of data at the transport layer.
SER	<i>See Satellite Equipment Room.</i>
server	A program on a computer that provides services to workstations. File, database, print, and communications are just a few kinds of servers.
server control procedure	Program used in Netwise RPC.
server header file	A text file generated by the Netwise RPC Compiler for the server that contains C language macro definitions, including directives and declarations. [Netwise RPC Tool]

---

server source file	A C source code file created by the Netwise RPC Compiler that includes server stub code, a dispatcher, and pack/unpack procedures. [Netwise RPC]
server stub	Source code generated by the Netwise RPC Compiler that allows a remote procedure to perform as if it were called locally. The programmer links one or more server stubs into the server program. [Netwise RPC Tool]
Service Access Point	The address for the user of a service.
Service Advertisement Protocol	NetWare protocol for publicizing the current network address of services.
service class	A concept in the LAT architecture to allow extensions for specific groups of applications, such as interactive terminals or windowing workstations.
Service Data Unit	The information a layer accepts from its client and sends over the network. Contrast with header or control information.
SESQUINET	<i>Sesquicentennial Network.</i> Texas-based regional network named for their sesquicentennial celebration. [RFC 1177]
session	Networking term used to refer to the logical stream of data flowing between two programs communicating over a network. Note that there are usually many different sessions originating from one particular node of a network.
set statement	A C language augmentation used in interface specifications to assign a value to an interface variable. [Netwise RPC Tool]
SFD	A message body type for X.400.
SFH	<i>Simple Forwarding Header.</i> A concept used in the Commercial Mail Relay (CMR) whereby a user of a commercial service sends to a designated mailbox with a simple header (e.g., the first message line containing the name of the destination user) which the gateway system uses for forwarding. See RFC 1168 for more details.

SG	<i>Study Group.</i>
SGI	<i>Silicon Graphics, Incorporated.</i>
SGML	<i>See Standard Generalized Markup Language.</i>
SGMP	<i>Simple Gateway Monitoring Protocol.</i> The predecessor to SNMP.
shell	A term that usually refers to the user interface on an operating system. On Unix systems, the C or Bourne shells are the primary user interfaces. Contrast with the kernel, which interacts with the computer at low levels.
SIA	<i>Stable Implementation Agreements.</i> The product of the OSI Implementors Workshops.
SIG	<i>Special Interest Group.</i>
SIGCOMM	<i>ACM Special Interest Group on Data Communication.</i> The professional society for people interested in data communications and networks.
Single Attachment Station	FDDI term for a station attached to the ring via a concentrator.
SLIP	<i>Serial Line IP.</i> A simple protocol for running IP over serial lines, defined in RFC 1055.
slot	An entry in a fixed-size table. Also a sublayer in the Local Area Transport architecture which allows multiple users to share a single datagram packet, each occupying a separate slot.
small moveable cabling system	IBM cabling system. Contrast with the large, nonmoveable system sold by the same company.
SMDS	<i>Switched Multi-megabit Data Service</i> A datagram-based public data network service developed by Bellcore and expected to be widely used by telephone companies as the basis for their data networks.
SMI	<i>Structure of Management Information.</i> Recommended Internet protocol defined in RFC 1155.



---

<b>S-mode</b>	Used in the OSI Virtual Terminal protocols. A model for terminal operation consisting of a two-way alternate dialogue.
<b>SMT</b>	<i>See station management.</i>
<b>SMTP</b>	<i>Simple Mail Transfer Protocol.</i> The Internet standard protocol for transferring electronic mail messages from one computer to another. SMTP specifies how two mail systems interact and the format of control messages they exchange to transfer mail. [RFC 1177]
<b>SNA</b>	<i>See System Network Architecture.</i>
<b>SNADS</b>	<i>SNA distribution services.</i> An architecture used for transferring messages in an SNA environment, similar to X.400.
<b>snail mail</b>	The traditional postal service.
<b>SNAP SAP</b>	<i>Subnetwork Access SAP.</i> A special form of Service Access Point where the first 5 bytes of the information field in the Logical Link Control data serve as the protocol identifier.
<b>Sniffer</b>	A network analyzer made by Network General. The Sniffer was used to produce the screen dumps of network packets in this book.
<b>Sniffer Network Analyzer</b>	Network General product used to monitor many different upper- and lower-layer network protocols.
<b>SNMP</b>	<i>Simple Network Management Protocol.</i> The standard network management protocol used in TCP/IP networks. Defined in RFC 1098.
<b>SNPA-address</b>	SubNetwork Point of Attachment address or an address at which the subnetwork service is available to the network entity. [RFC 1070]
<b>SOA</b>	<i>Start of Authority.</i> A marking in a DNS zone file indicating that this is the beginning of a new authority.
<b>socket</b>	An entry point to a program. User programs communicate with IPX by means of sockets. Each user typically has a separate socket.
<b>Softswitch</b>	Company that makes gateways between different message handling services.

SONET	<i>Synchronous Optical Network.</i> Fiber optic-based circuits.
SOSIP	<i>Swedish OSI Profile.</i>
source address	The origin of a data packet on a network.
SP	<i>Space.</i>
SPARC	<i>Scalable Processor Architecture.</i> A reduced instruction set (RISC) processor developed by Sun and licensed by several vendors including AT&T and Texas Instruments.
SPF	<i>Shortest Path First.</i> See OSPF.
spool	A place for a fast device (such as a software program) to leave data for later processing on a slow device (such as a printer).
SPP	<i>See Sequenced Packet Protocol.</i>
SQL	<i>See Structured Query Language.</i>
SRC	<i>Source address or Source From Address Correct.</i> Source address is the source data link address. Source From Address Correct is a bit in the HYPERchannel header that indicates that the 32-bit address in the from field is in fact the originating address for the message.
SS7	<i>Signalling System 7.</i> Protocol related to ISDN. Directs how the interior of an ISDN network is managed.
SSAP	<i>Source service access point.</i> Address of the user of a service. See also DSAP.
SSCP	<i>System services control point.</i> A network addressable unit in IBM's SNA architecture. Resides on a mainframe and is the central point for that domain of an SNA network.
SSS	<i>Server session socket.</i> AppleTalk Session Protocol field that contains the socket number to which session level packets are sent.
standard	A standard interface for two components. The nice thing about standards is there are so many to choose from.
Standard Control Interface	A simple, high-level programmatic interface to the Netwise RPC Server Control Library. [Netwise RPC Tool]

Standard Generalized Markup Language	ISO standard for the representation of revisable form text.
Standard Protocol	Classification for Internet standards. The IAB has established this as an official standard protocol for the Internet. These are separated into two groups: (1) IP protocol and above, protocols that apply to the whole Internet; and (2) network-specific protocols, generally specifications of how to do IP on particular types of networks. [RFC 1140]
standby monitor	Token ring term for a computer that is waiting for the active monitor to fail and is ready to step in if that happens. Kind of like a vice president.
Standby Monitor Present	Packet sent out by a standby monitor every 7 seconds to advertise its presence.
star coupler	Device used to connect different nodes of a VAX Cluster that use the CI bus.
state machine	A virtual machine that divides a process into logical subtasks called states which are organized and executed in a defined sequence. Netwise implements software state machines in the dispatch, the client stubs, and the server stubs. [Netwise RPC Tool]
station management	Part of the FDDI ring: the entity which monitors activity and exercises overall control.
STATSRV	<i>Statistics Server.</i> A network statistics protocol defined in RFC 996.
STE	<i>Signalling terminal exchange.</i> Equipment in an X.25 network that forms the boundary of a network. Communication between different X.25 management domains is between STEs using the X.75 protocols.
STN	<i>Switched Telephone Network.</i>
stored upstream address	Token ring concept. Each node on the token ring stores the address of the neighbor from which it receives data.
stream head	The entry point to a stream, a series of software modules connected with the STREAMS mechanism.

stream-oriented	A type of transport service that allows its client to send data in a continuous stream. The transport service will guarantee that all data will be delivered to the other end in the same order as sent and without duplicates. Also known as a reliable transport service.
STREAMS	An AT&T mechanism developed for the Unix operating system. STREAMS is a way of connecting a series of software modules, letting them send messages to each other.
structure file	A collection of agent and element structure definitions. [SunNet Manager]
Structured Query Language	International standard language for communicating with relational database systems.
stub	A piece of code that is used in RPCs. The stub looks like the called or calling procedure and thus masks the details of the RPC implementation from the calling or called procedures.
subnet	A term used to denote any networking technology that makes all nodes connected to it appear to be one hop away. In other words, the user of the subnet can communicate directly to all other nodes on the subnet. A subnet could be X.25, Ethernet, a token ring, ISDN, or a point-to-point link. A collection of subnets, together with a routing or network layer, combine to form a network.
subnet field	The bit field in an internet address denoting the subnet number. The bits making up this field are not necessarily contiguous in the address. [RFC 950]
subnetwork	A communications medium that provides a "direct" path between network layer entities. This can be realized via a point-to-point link, a LAN, a public data network, and so forth. It is worth noting that, unlike Internet Subnets, OSI Subnetworks are not necessarily reflected in the addressing hierarchy, so the double meaning of the Internet term "subnet" (a single IP hop; a part of the address hierarchy) does not hold in the OSI world. [RFC 1136]
subvector	Portion of a MAC frame. For example, the token ring command to request initialization on the ring contains subvectors with the adapter software level, upstream neighbor address, and several other fields.
Sun Microsystems	Makers of workstations and the Network File System.



---

<b>SURANET</b>	<i>Southeastern Universities Research Association Network.</i> An NSFnet regional network. [RFC 1177]
<b>SVC</b>	<i>See switched virtual circuit.</i>
<b>SVID</b>	<i>System V interface definition.</i> AT&T-sponsored definition used to determine the compatibility of different implementations of System V.
<b>switched virtual circuit</b>	A virtual circuit that is set up on demand, as in the case of a dial-up telephone line or an X.25 call. <i>See permanent virtual circuit.</i>
<b>SWS</b>	<i>Silly Window Syndrome.</i> A phenomenon found in TCP/IP whereby the available window is reduced to zero. Described by Dr. David Clark in RFC 813.
<b>symbol</b>	The smallest signalling element used at the MAC layer of the network. The symbol is then sent down to the physical layer for transmission. In FDDI, a symbol is five code bits at the physical layer.
<b>SYN</b>	<i>Synchronize.</i> A TCP bit that causes both ends to resynchronize the windows on a connection. Used to initiate or reset a connection.
<b>synchronization</b>	Coordination of tasks among multiple users. Synchronization mechanisms include locking and semaphores.
<b>synchronous</b>	An FDDI service class where each requester gets a preallocated maximum bandwidth and hence a guaranteed response time.
<b>synchronous processing</b>	A common method of operation wherein the calling procedure is suspended (blocked) and waits for a response after calling another procedure. [Netwise RPC Tool]
<b>system</b>	An instance of a component. Sometimes called a host or device. [SunNet Manager]
<b>System Network Architecture</b>	IBM's networking architecture.
<b>Systems Network Architecture</b>	IBM network architecture.

T1	A term for a digital carrier facility used to transmit a DS-1 formatted digital signal at 1.544 Mbps.
T3	A term for a digital carrier facility used to transmit a DS-3 formatted digital signal at 44.746 Mbps. [RFC 1177]
T.4	CCITT standard for Group 3 facsimile transmission.
T.6	CCITT standard for Group 4 facsimile transmission.
table	Relational database term for data grouped into rows and columns.
TAC	<i>Telephone Access Code.</i> A Milnet term.
tag qualifier	A C language augmentation to the declaration statement for transmitted variables and parameters. A tag qualifier tells the Netwise RPC Compiler to generate RPC code that uses the encoding rules for the tag qualifier type instead of using the rules for the declared type when creating a PDU. [Netwise RPC Tool]
target system	The host (or object) where a managed object resides. [Sun-Net Manager]
TC	<i>See transport connection.</i> Also means technical committee.
TCC	<i>Telephone Country Code.</i> Part of an X.121 address signifying the country being addressed.
TCP	<i>See Transmission Control Protocol.</i>
TCP/IP	<i>Transmission Control Protocol/Internet Protocol.</i> This is a common shorthand which refers to the suite of application and transport protocols which run over IP. These include FTP, Telnet, SMTP, and UDP (a transport layer protocol). [RFC 1177]
TCP-LDP	<i>TCP Extensions for Long Delay Paths.</i> Limited Use Internet protocol defined in RFC 1072
TDF	<i>Time Differential Factor.</i> The difference between the local time zone and Coordinated Universal Time (UTC).
Telenet	Packet switched network service offered by US Sprint.
Telex	Messaging mechanism that predates fax or electronic mail.

Telnet	The Internet standard protocol for remote terminal connection service. Telnet allows a user at one site to interact with a remote timesharing system at another site as if the user's terminal was connected directly to the remote computer. [RFC 1177]
TENEX	An early version of the Nicname or whois programs.
terminal emulator	A program that allows a computer to emulate a terminal. The workstation thus appears as a terminal to the host.
terminate and stay resident	DOS program that is loaded before application programs. The TSR programs are activated by a specific keystroke. Borland's Sidekick is an example of a TSR.
termination expression	Term used in the Netwise RPC Tool. The termination expression is included in structures when there is less data than the data structure could have held. The termination allows the RPC mechanism to send only the real data instead of transmitting blank cells for unused portions of the data structure.
terminator	Device on each end of an Ethernet cable to prevent reflections.
terrabite	One trillion bytes.
ThickWire	Another name for the 10BASE5 standard for coaxial cables and Ethernet.
ThinWire	Thinner, and cheaper, version of baseband coaxial cable used for Ethernet networks. Also called CheaperNet.
THT	<i>Token holding timer.</i> Token ring and FDDI term for the amount of time a node can transmit data before sending the token back out to the ring.
TID	<i>Transaction ID.</i>
TLI	<i>See Transport Level Interface.</i>
token bus	An alternative to token ring and Ethernet local area networks. Used in the MAP protocols. The token bus uses a multiple access protocol, but the device that "owns" the token is the only one that can send data.

token ring	A local area network protocol in which computers are connected together in a ring. A node waits until a token is passed around the ring, at which point it may send data. When it has finished sending, it releases the token and passes it to the next node. <i>See FDDI.</i>
topology	A network topology shows the computers and the links between them. A network layer must stay abreast of the current network topology to be able to route packets to their final destination.
TP	<i>Twisted Pair.</i>
TPDU	<i>Transport protocol data unit.</i> A packet of information exchanged between two transport layer entities in an OSI network. <i>See PDU.</i>
TR	<i>Technical Report.</i>
transaction	A series of one or more operations that form a logical whole. The entire transaction or none of it must take effect.
transceiver	A term used in Ethernet networks. The transceiver is the hardware device that connects to the Ethernet media, often a piece of coax cable. The transceiver is then connected to an Ethernet controller on the host system.
transceiver cable	A cable of up to 50 meters long between the transceiver and the Ethernet controller.
TransLAN	A wide-area extended Ethernet bridge manufactured by Vittalink.
Transmission Control Protocol	The transport protocol in TCP/IP used for the guaranteed delivery of data.
transport connection	Virtual channel of communication between two users provided by the transport layer module.
Transport Level Interface	AT&T-developed specification for the interface between the transport layer and upper-layer users.
transport service access point	The entry point to the transport module for a particular end user.
trap	Programming concept for a block of code that is executed whenever a specific condition occurs, usually an error.



---

traps statement	A C language augmentation that instructs the RPC Compiler to add custom error-handling instructions to the source code generated for the client or server stubs, or the dispatcher. [Netwise RPC Tool]
TRT	<i>Token rotation timer.</i> Token ring and FDDI term for the amount of time the token should take to go around the ring. <i>See also</i> TTRT.
trunk cable	Used to distinguish the coaxial Ethernet cable (the trunk) from the attachment to the individual node (the transceiver cable).
TSR	<i>See terminate and stay resident.</i>
TTL	<i>Time to Live.</i> Field in the Internet Protocol header which indicates how long this packet should be allowed to survive before being discarded.
TTRT	<i>Target token rotation time.</i> A term used in FDDI to set performance parameters. The TTRT serves as a measure of expected delay and is used, among other things, to set time-out parameters.
TTX	<i>Teletex.</i>
TTY	<i>Teletype.</i> A line-oriented terminal.
TUP	<i>Telephone User Part.</i> <i>See</i> Q.721.
tuple	A term used in relational database systems. A tuple is the equivalent of a record in a file management system and corresponds to one row of data in a table.
TW	<i>ThinWire.</i>
TWG	<i>The Wollongong Group.</i>
twisted pair	A pair of wires (or several pairs of wires) such as is used to connect telephones to distribution panels. Twisted pair is also being used as a physical transmission medium for Ethernet, token ring, and other forms of data links.
Tymnet	A public packet-switching network operated by McDonnell Douglas Network Systems Company. [RFC 1177]

Tymnet	Public packet-switched network based on X.25 owned by McDonnell-Douglas.
type definition	A statement that declares a user-defined type, including the struct, union, enum, typedef, or macrodef statements. [Netwise RPC Tool]
UA	<i>Unnumbered acknowledgment.</i> Type of frame used in the LLC operation.
UAM	<i>See User Authentication Method.</i>
UAPDU	<i>User Agent Protocol Data Unit.</i>
UCL	<i>University College London.</i> A United Kingdom research center which is quite active in the area of X.500 directories and X.400 message handling systems.
UDP	<i>See User Datagram Protocol.</i>
UI	<i>Unnumbered Information.</i> Type of frame used in the LLC operation.
UIC	<i>User identification code.</i> Number used to uniquely identify every user on the system.
Ultrix	Version of Unix sold by DEC for VAX computers.
Unibus	A peripheral bus used on 11/780 and 8600 VAX processors.
Unix	Operating system developed and trademarked by American Telephone and Telegraph. Unix is a pun on the Multics operating system.
unpack	Term used in remote procedure calls for translating data from the machine-independent form into the form used on a particular computer.
unpack procedure	A procedure created by the Netwise RPC Compiler that decodes a PDU from a standard format into C data types in the format for the host processor. [Netwise RPC Tool]
unsigned	A data type that has only positive numbers.
Update timestamp	An indication of when an entry was last updated.

---

USASCII	A subset of the ASCII character set corresponding to the characters below decimal 128.
USC	<i>University of Southern California.</i>
Usenet	Network of Unix users. This is a somewhat informal network of loosely coupled nodes that agree to exchange information in the form of electronic mail and a bulletin board.
User Authentication Method	AppleTalk concept for the way that users are identified to a file server before they can access resources.
User Datagram Protocol	Part of the TCP/IP protocol suite. UDP operates at the transport layer and, in contrast to TCP, does not guarantee the delivery of data.
UT	<i>Universal Time.</i>
UTC	<i>Coordinated Universal Time.</i> Coordinated international standard for time. Local time is the UTC plus a time differential factor.
UTnet	<i>University of Texas Network.</i>
UTS	<i>See Update timestamp.</i>
UUCP	<i>Unix-to-Unix copy program.</i> The standard Unix utility used to exchange information between any two Unix nodes. Used as the basis for Usenet.
V.21	CCITT standard for 300-bps duplex modem over the general switched telephone network.
V.22	CCITT standard for 1200-bps duplex operation over the general switched telephone network.
V.22 bis	CCITT standard for 2400-bps duplex modems over the general switched telephone network.
V.23	CCITT standard for 600- and 1200-bps modems.
V.24	CCITT standard for the definition of circuits between a DTE and DCE.
V.27	CCITT standard for 4800-bps modem over leased circuits.

V.27 bis	CCITT standard for 4800- and 2400-bps modem over leased telephone-type circuits.
V.27 ter	CCITT standard for 4800- and 2400-bps modem over general switched telephone networks.
V.29	CCITT standard for 9600-bps modem over four-wire leased telephone circuits.
V.32	CCITT standard for a family of two-wire modems operating up to 9600-bps over general and leased telephone circuits.
V.33	CCITT standard for 14.4-kbps modems over leased circuits.
V.35	CCITT physical interface standard for high-speed data transmission.
Value-Added Network	A network on top of a network.
Value-added reseller	Company that embeds another company's products into a more sophisticated product.
VAN	<i>See Value-Added Network.</i>
VAR	<i>See Value-added reseller.</i>
variable mode	<i>See external variable mode.</i>
VAX	<i>Virtual address extension.</i> Hardware series made by DEC.
VAX Clusters	DEC Clusters that operate on the CI bus rather than over the Ethernet.
VCN	<i>See Virtual Block Number.</i>
verify function	Agent routine that checks the validity of a request before the request is dispatched. [SunNet Manager]
view	A collection of components in the Console's MDB. Views have members, called elements. Since views are elements, they may be members of other views. Elements may have multiple memberships. Memberships may be circular. [SunNet Manager] Also a database term that refers to making a collection of one or more tables appear as a single database table.



VINES	Network architecture made by Banyan.
virtual circuit	A service offered usually at the transport layer. The user of a virtual circuit is able to send data to a remote user and not worry about putting data in packets, error recovery, missing data, or routing decisions.
virtual memory	An operating system concept that refers to the address space of a program. A paging system maps virtual memory to the limited physical memory pages as needed.
Virtual Terminal	An OSI service that allows a user on one system to log onto another for an interactive session.
VM	<i>Virtual machine or see virtual memory.</i> VM is an IBM operating system that permits guest operating systems, such as MVS, to reside on top of it. Usually used in conjunction with the CMS user interface.
VMS	<i>Virtual memory system.</i> A DEC proprietary operating system for VAX computers.
VMTP	<i>Versatile Message Transaction Protocol.</i> A transport layer protocol defined in RFC 1045.
VSAM	<i>Virtual sequential access method.</i> File organization method used in IBM environments for direct access files. Similar to ISAM (indexed sequential access method).
VT	<i>See Virtual Terminal.</i>
VT100	A series of DEC terminals.
VTAM	<i>Virtual Telecommunications Access Method.</i> An IBM software system that provides the interface to an SNA network.
WAN	<i>Wide-area network.</i> Sometimes also used to mean work area network or a small subnetwork for a work group.
WESTNET	One of the National Science Foundation-funded regional TCP/IP networks that covers the states of Arizona, Colorado, New Mexico, Utah, and Wyoming. [RFC 1177]

WG	<i>Working Group.</i>
whois	An Internet program which allows users to query a database of people and other Internet entities, such as domains, networks, and hosts, kept at the NIC. The information for people shows a person's company name, address, phone number, and email address. [RFC 1177]
WIA	<i>Working Implementation Agreement.</i> Temporary agreements not having reached the status of a Stable Implementation Agreement (SIA).
wide-area bridge	A MAC-layer bridge that works on wide-area communications links such as T1 and dial-up lines.
window	The amount of data that can be outstanding and unacknowledged on a given TCP connection.
WKS	<i>Well-known service.</i> A service on TCP or UDP that uses a well-known port number and can thus be accessed without a priori knowledge of which port the application may be using at a given time.
workgroup	Trendy term for people who work together. Several computers may be isolated on a small network, known as a workgroup network. Whether anything is accomplished there is another matter.
WORM	<i>Write once/read many.</i> A type of optical disk that can be written locally, contrasted to CD-ROM disk. Also used for fishing.
write lock	Also known as an exclusive lock. Prevents others from reading or writing the locked data.
WRL	<i>See Western Research Laboratory.</i>
WRQ	<i>Write Request.</i> Packet type in the Trivial File Transfer Protocol (RFC 783).
WYSIWYG	<i>What You See is What You Get.</i>
X.3	CCITT standard for a packet assembler/disassembler (PAD).
X.12	ANSI committee for Electronic Data Interchange
X.21	CCITT standard for circuit-switched networks.

---

X.21 <i>bis</i>	Use of the synchronous V-series modems over public data networks.
X.25	CCITT standard for the interface between a DTE and DCE for terminals operating in packet mode and connected to the public data network with a dedicated circuit.
X.28	CCITT protocols for an asynchronous terminal to communicate with an X.3 PAD.
X.29	CCITT protocols for a synchronous DTE (a host) to control and communicate with an X.3 PAD.
X.75	CCITT standard for interconnecting separate X.25 networks.
X.81	Internetworking between an ISDN circuit-switched network and a circuit-switched public network.
X.110	CCITT standard for routing principles on public data networks.
X.121	CCITT numbering plan for public data networks.
X.200	CCITT version of the OSI reference model.
X.208	CCITT version of the OSI ASN.1
X.209	CCITT version of the OSI ASN.1 Basic Encoding Rules (BER).
X.211	Physical service definition for OSI for CCITT applications.
X.212	Data link service definition for OSI for CCITT applications.
X.213	Network layer service definition for OSI for CCITT applications.
X.214	Transport service definition for OSI for CCITT applications.
X.215	Session service definition for OSI for CCITT applications.
X.216	Presentation service definition for OSI for CCITT applications.
X.217	ACSE definition for OSI for CCITT applications.
X.218	CCITT equivalent of ISO 9066-1: Text communication—reliable transfer.

- X.219 CCITT equivalent of the ISO Remote Operations Service Element (ROSE).
- X.220 CCITT specification of the use of X.200-series protocols in CCITT applications.
- X.223 Use of X.25 to provide the OSI connection-mode network service.
- X.400 CCITT standard for message-handling.
- X.402 CCITT message handling system: overall architecture.
- X.403 CCITT message handling system: conformance testing.
- X.407 CCITT message handling system: abstract service definition conventions.
- X.408 CCITT message handling system: encoded information type conversion rules.
- X.411 CCITT message handling system: message transfer system: abstract service definition and procedures.
- X.413 CCITT message handling system: message store, abstract-service definition.
- X.419 CCITT message handling system: protocol specifications.
- X.420 CCITT message handling system: interpersonal messaging system.
- X.500 CCITT standard for directory information.
- X.501 CCITT directory: models.
- X.509 CCITT directory: authentication framework.
- X.511 CCITT directory: abstract service definition.
- X.519 CCITT directory: protocol specifications.
- X.520 CCITT directory: selected attribute types.
- X.521 CCITT directory: selected object classes.



<b>XDR</b>	<i>See External Data Representation.</i>
<b>Xerox</b>	Maker of a variety of computer and imaging products. Inventor of an even wider variety of products.
<b>Xerox Network System</b>	A set of upper-layer (layers 3 and 4) protocols, plus some applications, typically used in conjunction with Ethernet. An alternative to DECnet or TCP/IP. The layer 3 and 4 protocols form the basis for Novell's NetWare.
<b>Xerox Palo Alto Research Center</b>	Xerox research laboratory that led to the development of many current technologies including PostScript, the Apple Macintosh, and Ethernet.
<b>XID</b>	<i>Exchange identification.</i> An HDLC frame used when a new node attaches to the physical medium. The XID frame contains information such as the node ID or a verification password for the connection.
<b>XMI</b>	A 100-Mbps bus used to connect CPUs, BI buses and memory on the VAX 6200 series of parallel processors.
<b>XNS</b>	<i>See Xerox Network System.</i>
<b>XNS Mail Transport Protocol</b>	Message handling service in XNS.
<b>Yellow Pages</b>	A set of services in the Network File System that propagate information out from masters to recipients. Used for the maintenance of system files on complex networks. Yellow Pages are known in marketing-speak as the Network Information Services.



# Index

- Abstract Syntax Notation One (ASN.1), 193, 468
- Access control, 152-53, 463
- Access rights, 236, 425
- Accountability, 262
- Acknowledge (ACK), 76-83, 86, 463
- Activity daemon, 464
- Activity requests, 421
- Adaptive retransmission, 163
- Address, 464
  - DNS, 252-54
  - IP, 63-64, 67-70
  - RIP, 94
  - subnetworks, 14, 33
- Address (A) record, 338
- Address Resolution Protocol (ARP), 16, 62, 70, 464
- Administration Management
  - Domain (ADMD), 347, 464
- Agent, 407, 465
- Agent library, 416, 425
- Agent/manager model, 407
- Alarm, 420
- Amateur Packet Radio Network (AMPRnet), 54
- Ampersand, 290
- AMPRnet. *See* Amateur Packet Radio Network
- Anonymous FTP, 274
- AppleTalk, 457, 467
- Applications, distributed. *See* Distributed applications
- Architecture
  - interoperability, 435
  - license service, 297-300
  - management, 408, 416, 420
  - PC-NFS, 318-20
- A record. *See* Address (A) record
- ARP. *See* Address Resolution Protocol
- ARPANET, 105-106, 239
- ASA format control, 382
- ASCII codes, 382, 386, 468
  - see also* NetASCII
- ASN.1. *See* Abstract Syntax Notation One
- AS routing, 100, 101
- Associations, 204-205
- Asynchronous mechanisms, 140-41, 469
- Asynchronous processing, 193, 469
- Asynchronous transmission, 29
- Atomic updates, 304-305
- Attributes, 170, 172, 408, 470
- Auspex, 173-77
- Authentication, 470
  - NTP, 209-11
  - security, 262, 264, 265, 271-72, 276
- Authorization, 262, 470
- Automounter, 42, 153, 281-92
  - etc/fstab, 281-82
  - maps, 285-92
  - multiple mounts and locations, 287-90
- NFS, 284-85
  - overview, 282-84
  - subdirectory field, 290

- substitution, 290
- use, 292
- AX.25 protocol, 53-54
- Background mount, 155
- Badarea mailer, 347
- Balun, 18, 471
- Bandwidth, 85, 471
- BARRnet. *See* Bay Area Research Network
- Basic Encoding Rules (BER), 193, 409, 411
- Bay Area Research Network (BARRnet), 42-44
- BER. *See* Basic Encoding Rules
- Berkeley Internet Name Domain (BIND), 240
- Berkeley routed daemon, 97
- BGP. *See* Border Gateway Protocol
- Binary data set, 170
- BIND. *See* Berkeley Internet Name Domain
- Binding, 187-88, 224, 472
- BIOD. *See* Block I/O Daemon
- Bisbey, Richard, 54
- BISDN. *See* Broadband ISDN
- Bisynch (BSC), 440, 473
- Block I/O Daemon (BIOD), 163-66
- Block mode, 384
- Bootling, 275-76, 396
- Boot parameters (BOOTP), 68, 70, 474
- Border Gateway Protocol (BGP), 95, 100
- Braden, Robert, 85
- Bridges, 21-23, 474
- Broadband ISDN (BISDN), 36, 474
- Broadcasts, 22, 95, 139-140, 224, 319, 474
- BSC. *See* Bisynch
- Byte size, 382
- Caching, 168, 243, 255, 321, 476
- Calendar tool, 147
- Callback RPC, 140-41
- Canonical name (CNAME), 338, 339
- Carrier Sense, 13, 476
- CASA network, 50, 53, 477
- Cd command. *See* Change directory command
- CERFnet, 46, 50
- CG3270 software, 439-40, 444
- Change directory (cd) command, 287, 288
- Channel gateways, 436-38, 478
- Character generator, 400
- Clark, David, 82
- Client stub, 183, 184, 186, 188-89, 192, 480
- CLNP. *See* Connectionless Network Protocol
- CLNS. *See* Connectionless Network Service
- Clock synchronization, 200-204, 206-208
- Clouds, 426
- Cluster records, 419, 480
- CMIP language, 408, 480
- CMS operating system, 173, 481
- CNAME. *See* Canonical name
- Common Data Format (NASA), 144-45
- Compilers, 181-85
- Compressed mode, 384
- Concurrent access, 172-73, 482
- Configuration files, 107
- Configuration rule, 19
- Congestion, 87-88, 482
- Connection, 78, 84-88, 111-12, 188-89, 482
- Connectionless Network Protocol (CLNP), 68-69
- Connectionless Network Service (CLNS), 61, 88, 480
- Connection-Oriented Network Service (CONS), 88, 482
- ConneXions, 111
- CONS. *See* Connection-Oriented Network Service, 88
- Content type indication, 350
- Control-unit emulation, 437, 483
- Conversation key, 265-66
- Cookie, 264, 308, 483
- Coordinated Universal Time (UTC), 200, 201, 555
- Create access, 235-36
- Credentials, 264



- Cryptography, 262, 264-66
- CSMA/CD protocols, 12-13, 18
- C2 security option, 275
- Customization, 192-93
- Daemons, 485
  - activity, 421, 464
  - Automounter, 287
  - license service, 302
  - locking, 321
  - networks, 97, 107
  - NFS, 151-52, 163
  - NIS, 219
  - PC-NFS, 320-22
  - printer, 400-404, 446
  - RPC, 134
- Database synchronization, 100, 107
- Data Encryption Standard (DES),
  - 264, 265, 270, 487
- Data ingester, 144
- Data link, 61, 490
- Data mark (DM), 373
- Data packet, 78
- Data records, 420-21
- Data representation, 382-84
- Data set, 170-71
- Data transfer process (DTP), 382
- Data types, 186, 411
- Data unit, 387
- Daytime protocol, 400, 485
- DDN. *See* Defense Data Network
- DEC systems, 19, 306, 408, 436,
  - 452-54, 487
- Default maps, 219
- Defaults, 95, 104, 171, 487
- Defense Data Network (DDN),
  - 105-106, 486
- DES. *See* Data Encryption Standard
- Destination host, 64
- Destroy access, 235-36
- Device driver, 120
- Diffie-Hellman method, 266-67
- Direct map, 285-86
- Director, 407, 488
- Directory, 229, 233, 488
- Direct programming, 130
- Discard protocol, 400, 489
- Discover Tool, 421-23, 427
- DISOSS, 441, 489
- Dispersion number, 204
- Distance vector algorithms, 92-93
- Distributed applications, 295-322, 489
  - license service, 296-303
  - network lock manager protocol,
    - 303-309
  - overview, 295-96
- Distributed environment, 182, 183
- Distribution lists, 360-63, 490
- DM. *See* Data mark
- Domain Name System (DNS), 112,
  - 218, 231-32, 239-56, 490
  - addresses, 252-54
  - caching, 255
  - configuration, 243-45
  - implementation, 243
  - message compression, 255-56
  - message forwarding, 338-39
  - namespace, 240-42
  - operation, 249-56
  - recursion, 254-55
  - resource records, 245-49
  - security, 262
  - servers, 240
  - zones, 242
- Drivers, 318-19
- DTP. *See* Data transfer process
- EBCDIC codes, 382, 492
- Echoes, 74, 86, 373, 492
- EDS. *See* Electronic Data Systems
- EEPROM file, 236-37
- EGP. *See* Exterior Gateway Protocol
- Electronic Data Systems (EDS), 169
- Encoding, 409
- Encryption, 262, 277
- End of file (EOF), 383-84, 494
- Entry procedures, 192
- EOF. *See* End of file
- Epoch systems, 175-77
- Errors, 69, 412, 494
- Etc/fstab file, 281-82
- Etc/hosts file, 222, 263
- Etc/passwd file, 220

- Ethernet, 11-31, 495
  - bridging, 21-23
  - configuration, 16-18, 19
  - IEEE LANs, 15-20
  - multisegment, 19-20
  - NFS, 174
  - packet formats, 13-14
  - PC-NFS, 319
  - protocols, 12-13
  - traffic, 16
- Events, 420, 428, 495
- Exchange identification (XID), 16, 561
- Exclusive lock. *See* Write lock
- Exit procedures, 192
- Expand (EXPN) command, 330
- Expansion point, 360-61
- EXPN command. *See* Expand (EXPN) command
- Export file system (exportfs), 151, 152-53, 168
- Extension fields, 334
- Exterior Gateway Protocol (EGP), 62, 95, 100-105
- External Data Representation (XDR), 144-46, 417, 496
- FADUs. *See* File access data units
- Fax Mail, 41, 498
- FCS. *See* Frame check sequence, 14
- FDDI. *See* Fiber Distributed Data Interface
- FEP. *See* Front End Processor
- Fiber, 18
- Fiber Distributed Data Interface (FDDI), 24-30, 50, 93, 498
- Fiber repeater, 19
- File access, 168, 499
- File access data units (FADUs), 387
- File handle. *See* Cookie
- File processor, 174
- File sharing, 309
- File structure, 383
- File system, 371-404, 499
  - FTAM, 377-92
  - FTP, 377-86
  - resource location, 393-96
  - table, 281
- File Transfer, Access, and Management (FTAM), 377-92, 500
  - document types, 388-90
  - functional units, 387-88
  - implementation, 390-93
  - service classes, 387
- File Transfer Protocol (FTP), 78, 239, 274, 377-86, 500
  - commands, 380-81
  - data representation, 382-84
  - traffic, 384-86
  - see also* Trivial File Transfer Protocol
- Filtering, 22, 95
- Find state, 188
- Finger protocol, 394-96, 499
- Finish state, 189
- Fixed metric, 93
- Flow control, 80
- Forwarding, 243, 334-37
- Forward path, 329
- Fragmentation, 65
- Frame, 27, 499
- Frame check sequence (FCS), 14
- Frequency synchronization, 200
- Front End Processor (FEP), 169
- FTAM. *See* File Transfer@ Access@ and Management
- FTP. *See* File Transfer Protocol; Trivial File Transfer Protocol
- Functional units, 388
- Garbage collection timer, 95
- Gateways, 500
  - interoperability, 436-38, 440
  - mail, 364-65
  - NFS, 169
  - subnetworks, 65, 95
  - TCP, 74, 83-84, 88
- Get request, 412
- Get state, 189
- GID. *See* Group ID
- Global tree, 409
- Global variables, 186, 501
- Glyph, 418, 412, 501
- Graphical User Interface (GUI), 193

- Graphics files, 144
- Group, 234, 502
- Group address, 14
- Group ID (GID), 264
- Group operations, 235
- GUI. *See* Graphical User Interface
- Hagens, Rob, 68
- Ham radio, 53-57
- Hardcoding, 281
- Hard mount, 153
- Hardware implementations, 173-77
- HDL. *See* High-Level Data Link Control
- Header fields, 334
- HELLO Protocol, 99-100, 104, 502
- Hesiod, 270
- High-Level Data Link Control (HDL), 33, 34
- High Performance Parallel Interface (HIPPI), 50, 503
- High Speed Serial Interface (HSI), 41
- HIPPI. *See* High Performance Parallel Interface
- Hooks, 192, 503
- Hops, 92, 93, 504
- Host, 222, 504
- HSI. *See* High Speed Serial Interface, 41
- HYPERchannel, 44, 50
- I. *See* Information
- IAC. *See* Interpret As Command
- IBM mainframe, 169-73
- IBM SNA. *See* System Network Architecture
- ICMP. *See* Internet Control Message Protocol
- Idempotent operations, 166, 305, 505
- IDs, 13-14, 65, 167, 264
- IEEE LANs, 15-20, 506
- Ifconfig command, 106-107, 431
- IGP. *See* Internal Gateway Protocol
- I-Heard-You (IHU), 100, 104
- IMessagesLink command, 123
- IN-ADDR.ARPA Domain, 252-54
- Index, 411, 507
- Indirect map, 285, 286-87
- INETD. *See* Internet Services Daemon
- Infinite Storage, 175
- Information (I), 54
- Information Resources (IR), 42
- Initial sequence number (ISN), 76
- Input/output processing, 118-19, 507
- In-Reply-To header, 331
- Instability, 83-84
- Internal Gateway Protocol (IGP), 97, 506
- International Atomic Time (TAI), 200-201
- International Organization for Standardization (ISO), 88, 193, 508
  - see also* ISO Development Environment
- International Telecommunications Union (ITU), 347
- Internet, 36, 41, 68, 212, 261, 325, 508
- Internet Control Message Protocol (ICMP), 70-74, 200, 213, 505
- Internet Protocol (IP), 10, 33, 61, 62-70, 84, 200, 261, 509
- Internet Router with a SunLink Channel Adapter (IRCA), 442, 444-45
- Internet Services Daemon (INETD), 107, 302
- Internetworks, 36, 509
- INTEROP, 412
- Interoperability, 435-58
  - case studies, 442-52
  - DEC, 436, 452-54
  - OSI, 454-55
  - overview, 435-36
  - PC, 436, 448-49, 455-57
  - SNA, 435-52
  - SunLink, 436-42
- Interpersonal Messaging Service (IPMS), 349, 350-53, 510
- Interpret As Command (IAC), 373
- Interprocess communication (IPC), 182
- IP. *See* Internet Protocol
- IP Control Protocol (IPCP), 33
- IPC. *See* Interprocess communication

- IPMS. *See* Interpersonal Messaging Service
- IR. *See* Information Resources
- IRCA. *See* Internet Router with a SunLink Channel Adapter
- IS-IS, 98
- ISN. *See* Initial sequence number
- ISO. *See* International Organization for Standardization
- ISO Development Environment (ISODE), 89, 510
- ISTREAMSPOP option, 120
- ISTREAMSPUSH argument, 120
- ITU. *See* International Telecommunications Union
- Jacobson, Van, 85, 88
- Jain, Raj, 88
- Juszczak, Chet, 167
- KA9Q Internet Protocol Package, 54
- Karn, Phil, 32, 53, 54
- Kerberos, 264, 273, 512
- LAM. *See* License Access Module
- LAN technology, 15, 512
- LAPB. *See* Link Access Procedure B
- LAT. *See* License Administration Tool
- LFN. *See* Long, fat pipe network
- License Access Module (LAM), 302
- License administration, 297, 302-303
- License Administration Tool (LAT), 302
- License library, 297, 298, 301-302
- License production tool, 297, 302-303
- License server, 297
- License service, 295, 296-303
- architecture, 297-300
  - license, 300
  - operation, 301-303
- Line printer (Lpr), 400-404
- Link Access Procedure B (LAPB), 54
- Link Quality Monitoring (LQM), 33
- Link state advertisement (LSA), 99
- LLC. *See* Logical link control
- Locking, 303-309, 321, 514
- Lock manager, 295, 306-309, 513
- Lock release, 307-308
- Lock request, 308
- Logical byte size, 382
- Logical link control (LLC), 15, 16, 514
- Long, fat pipe network (LFN), 85, 513
- Lookup operation, 162
- Loops, 372
- Lpr. *See* Line printer
- LQM. *See* Link Quality Monitoring
- LSA. *See* Link state advertisement
- Lynch, Dan, 418
- MX record. *See* Mail exchange (MX) record
- MCI Mail, 364, 516
- MAC. *See* MacIntosh computers; Medium Access Control
- MacIntosh computers, 457, 514
- MAC-layer bridge, 9
- Mail
- X.400,
    - messages, 349
    - munging, 365-66
    - naming, 349-50
- Mail, 325-67, 515, X.400 347-50, 365-66
- case study, 342-47
  - distribution lists, 360-363
  - DNS, 338-39
  - gateways, 364-65
  - IPMS, 350-53
  - message stores, 363-64
  - message transfer agents, 353-60
  - munging messages, 341, 365-66
  - overview, 325
  - RFC 822, 331-37
    - forwarding, 334-37, 338-39
    - from, sender, reply to, 337
    - message encapsulation, 337-38
  - RPC, 194
  - SMTP, 325-31
    - paths, 330-31
    - users, 328-30
  - subnetworks, 41
  - UUCP, 339-42
  - XDR, 141, 147



- Mailbridge, 105-106, 515
- Mail exchange (MX) record, 250-51, 338
- Mainframe access, 41
- Maintenance commands, 219
- Makedbm program, 219
- Management, 407-31
  - database, 418-21, 516
  - Discover Tool, 421-23
  - names, 409-412
  - operation, 426-29
  - SNMP, 407-408, 412-15
    - security, 425-26
    - support, 423-26
  - structure, 409
  - tools, 429-31
  - see also* SunNet Manager
- Management information base (MIB), 408, 425, 517
- Manager, 407, 409, 515
- Manager Services, 415, 515
- MANs. *See* Metropolitan area networks
- Maps
  - Automounter, 285-92
  - built-in, 291
  - indirect, 286-87
  - interoperability, 440
  - NFS, 173
  - NIS, 217-218, 219-22
- Marshall, Greg, 111
- Master map, 285, 286
- Master server, 222-26, 243
- MAU. *See* Multistation access unit
- Maximum transfer unit (MTU), 65, 82, 84, 86, 140, 163, 519
- MCP. *See* Multiprotocol Communications Processor
- Medium Access Control (MAC), 15, 516
- Messages
  - encapsulation, 337-38
  - flags, 359
  - forwarding, 338-39
  - munging, 341-42, 365-66
  - names, 255-56
  - NTP, 201-204
  - SMTP, 325-31
  - stores, 363-64
  - STREAMS, 117, 121-23
- Message transfer agents (MTA), 353-60, 517
- Metropolitam area networks (MANs), 35
- MIB. *See* Management information base
- Military network, 105-106, 517
- Mills, David, 54, 199, 212
- Milnet. *See* Military network
- Modification time, 162
- Modify access, 235-36
- Mount, 153-57, 171, 281-84, 518
- Mount points and options, 285-89
- MS-DOS, 308-309
- MTA. *See* Message Transfer Agents
- MTU. *See* Maximum transfer unit
- Multiaccess networks, 99
- Multiple Access, 13
- Multiple-client control procedure, 190
- Multiple mounts and locations, 287-90
- Multiport transceiver, 18, 520
- Multiprotocol Communications Processor (MCP), 41, 516
- Multistation access unit, 24-25
- Multi-tasking control procedure, 190
- Multithreaded server control procedure, 191-92, 520
- Munging, 341-42, 366-67
- MVS operating system, 169-73
- Names, 217-57
  - addresses, 252-54
  - combining services, 256
  - DNS, 239-56
  - mail, 349-50
  - management, 409-12
  - maps, 219-222
  - NFS, 173
  - NIS, 217-28
  - operations, 234-35, 249-56
  - resource records, 245-49

- RPC, 188
  - security, 235-37
  - servers, 240
  - see also* Domain Name System; Network Information Service; NIS+
- NAS. *See* Numerical Aerodynamic Simulation
- Negotiate About Window Size (NAWS), 376
- Negotiated options, 372
- Neighbor acquisition, 101, 104
- NetASCII, 382, 384
- NetCDF, 144
- NetISAM, 302
- Netstat command, 107-108
- NetWare, 457, 522
- Netwise RPC tool, 181, 182, 193, 522
- Network File System (NFS), 3, 41, 44, 90, 130, 151-77, 523, 524
  - Automounter, 281, 284-85
  - components, 151-52
  - concurrent access, 172-73
  - exports, 151, 152-53
  - file server, 173
  - hardware, 173-77
  - implementation, 163-69, 173-77
  - interoperability, 435, 449, 453
  - management database, 418
  - markets, 176-77
  - mounting, 153-54
  - MVS, 169-73
  - PC, 315-22, 320
  - protocol, 157-63
  - remote booting, 275-76
  - root access, 267-68
  - security, 264, 267-69, 275-77, 311, 425-26
  - server task, 170-72
  - state, 166-69
  - storage, 175-76
  - VM/CMS operating system, 173
- Network Information Center (NIC), 239, 524
- Network Information Service (NIS), 3-4, 217-28, 523
  - binding, 224
  - components, 219
  - definition, 217-19
  - limitations, 226-28
  - maps, 217-18, 219-22
  - RPC, 130
  - servers, 218-20, 222-26
  - starting up, 225-26
  - subnetworks, 42
  - TCP, 106, 393
  - see also* Domain Name System; NIS+
- Network library, 183-84, 319, 523
- Network lock manager protocol, 303-309
- Network management protocol, 407
- Networks, 61-113, 522
  - ARP, 69-70
  - CLNP, 68-69
  - connections and options, 84-89, 111-12
  - defined, 11, 61-62, 65, 522
  - EGP, 100-105
  - ICMP, 69-74
  - interoperability, 435-58
  - IP, 63-69
    - addressing, 63-64, 67, 522
    - fragmentation, 65
    - layers, 68-69
    - operation, 64-65, 106-11
    - subnetworks, 65-68
  - IS-IS, 97-100
  - management, 407-31
  - NSFnet, 105-106
  - OSPF, 97-100
  - RIP, 90-97
    - requests, 95-96
    - response processing, 96-97
  - TCP, 74-89
    - acknowledgment, 80-81
    - extending, 87
    - OSI, 88-89

- routing, 83-84
- silly window syndrome, 82-83
- source quench, 87-88
- SUN, 106-11
- UDP, 90
- see also* Interoperability;
  - Management; Network File System; Network Information Service; Network Time Protocol; NIS+
- Network Time Protocol (NTP),
  - 199-213, 400
  - accuracy, 213
  - associations, 204-205
  - control messages, 208
  - functions, 200-201
  - implementation, 212-13
  - message exchange, 201-204
  - protocol, 206-208
  - warped servers, 208-12
- Network virtual terminal (NVT),
  - 372-74
- NFS. *See* Network File System
- NFS Server Daemons, 163-66
- NIC. *See* Network Information Center
- NIS. *See* Network Information Services; NIS+
- NIS+, 226-39
  - backward compatibility, 239
  - names, 229-32
  - namespace replication, 237-38
  - objects, 232-34
  - security, 235-39
  - services, 234-35
- Nodes, 13, 22, 82, 524
- Nonblocking, 140
- Nonidempotent requests, 166-67
- Nonpersistent binding, 187, 525
- NSFnet, 43-44, 46, 50, 105-106, 526
- NTP. *See* Network Time Protocol
- Numerical Aerodynamic Simulation (NAS), 44, 155
- Objects, 229-30, 232-34, 409, 412, 418
- Obsolete status, 412
- Offered window, 82
- Offset, 201-204, 206-208
- ONC. *See* Open Network Computing
- ONC lock manager, 306-309
- Open Network Computing (ONC), 3,
  - 130, 193, 527
- Open Shortest Path First (OSPF),
  - 97-100, 528
- OpenWindows, 196, 415
- Options, 65, 84-88, 372, 373, 374-75
- O/R address. *See*
  - Originator/recipient address
- Originator/recipient (O/R) address,
  - 349, 527
- OSI Connectionless Network Service,
  - 61, 68, 386, 454-55
- OSPF. *See* Open Shortest Path First
- Output control window, 440
- Output discarding, 375-376
- Packet, 319, 528
- Packet assembler/disassembler (PAD), 34, 528
- Packet radio, 53-57
- Packing data, 186
- Pack state, 189
- PAD. *See* Packet assembler/disassembler
- Paging, 383, 529
- Parameter problem message, 73
- Password, 220-22, 225, 231, 261-77
  - see also* Security
- Paths, 330-31, 529
- PC. *See* PC-NFS; PC-NFSD; Personal computer]
- PC-NFS, 295-96, 315-22
  - architecture, 318-20
  - daemon, 320-22
  - locking, 321
  - printing, 321-22
- PC-NFSD, 321-22
- PC-NFS Programmer's Toolkit (PTK),
  - 315-17
- Pcnfs.sys driver, 318-19
- PDU. *See* Protocol data unit
- Peer-to-Peer software, 441
- Personal computer (PC), 315-22, 529

- interoperability, 436, 447-48, 455-57
- mail, 329, 363-64
  - see also* PC-NFS
- PI. *See* Protocol interpreter
- PID. *See* Protocol ID
- Ping command, 430
- Pipe, 85, 86
- Pointer, 233-34, 531
- Point-to-Point Protocol (PPP), 11, 32-34
- Polling rates, 212
- Port, 74
- Portmapper, 134-40
- Postal delivery system, 364, 530
- Postel, 398
- PPP. *See* Point-to-Point Protocol
- Primary time server, 201, 212
- Principals, 229
- Printers, 321-22, 371-404
  - daemon, 400-404, 446
  - interoperability, 446
- Private Management Domain (PRMD), 347
- Procedure declarations, 185-87
- Process binding, 187-88
- Programming, direct, 130
- Protocol data unit (PDU), 16, 189, 533
- Protocol ID (PID), 54, 531
- Protocol interpreter (PI), 377, 382
- Protocol messages, 120-21
- Protocol stacks, 117-18, 533
- Proxy agents, 417, 419, 533
- PTK. *See* PC-NFS Programmer's Toolkit
- PutMessagesnext procedure, 121
- Quality of Service (QOS), 69, 535
- Queues, 121
- Quote server, 42
- R\* commands, 295, 313-15
- Radio, amateur, 53-57
- RAID. *See* Redundant Array of Intelligent Disks
- RARP. *See* Reverse Address Resolution Protocol
- Rcp. *See* Remote copy program
- Read access, 235-36
- Read locks, 304
- Read request, 162
- Record structure, 383
- Recursion, 254-56, 361
- Redirect message, 73-74
- Redundant Array of Intelligent Disks (RAID), 50
- Remote booting, 275-76, 396
- Remote copy program, 314
- Remote Execution (REX) service, 295, 309-13
- Remote file systems, 281
- Remote log-in, 314
- Remote Procedure Call (RPC), 42, 107, 129-41, 536
  - asynchronous, 140-41
  - example, 146-48
  - library access, 131-34
  - NFS, 162, 169-70
  - Portmapper, 134-40
    - broadcast, 139-40
    - transport, 137-39
  - REX, 310, 311
  - security, 262, 263-73, 276-77, 311, 425
  - see also* Remote Procedure Call Tool
- Remote Procedure Call (RPC) Tool, 181-96, 539
  - client stub, 188-89
  - compilers, 181-85, 539
  - customization, 192-93
  - interfaces, 193-96
  - ISO, 193
  - Netwise, 182-85
  - procedure declarations, 185-87
  - process binding, 187-88
  - server control procedure, 189-92, 529
- Remote user information program (RUIP), 394-95, 540
- Renaissance service, 176
- Repeater, 19, 21, 536
- Replica operations, 235



- Reply-To field, 334, 337
- Requests, 95-96, 536
- Resource location, 393-98
- Resource records (RRs), 245-49, 338, 339
- Response processing, 96-97
- Restricted mode, 29, 537
- Retransmission problems, 166
- Return-Path, 334
- Reverse Address Resolution Protocol (RARP), 70
- REX facility. *See* Remote Execution (REX) service
- RFC 822, 331-37, 366
  - forwarding, 334-37
  - messages encapsulation, 337-38
  - Reply-To field, 337
- RFC 1091, 376
- RFC 1096, 377
- RFC 1155, 409
- Ring initialization, 30
- RIP. *See* Routing Information Protocol
- Rlogin. *See* Remote log-in
- Romkey, John, 418
- Root, 152, 229, 267-68, 275, 538
- Rose, Marshall, 89
- Round-trip delay, 85, 201, 206
- Round-trip timing (RTT), 86
- Routers, 9, 18, 61-62, 83-84, 90-97, 100-106, 538
- Routing Information Protocol (RIP), 90-97, 538
- RPC. *See* Remote Procedure Call
- RPC Tool. *See* Remote Procedure Call Tool
- RRs. *See* Resource records
- RSA method, 266
- Rsh command, 309, 312, 314
- RTT. *See* Round trip timing
- RUIP. *See* Remote user information program
- Rwho command, 314-15
- SACK. *See* Selective acknowledgment
- San Diego Supercomputer Center (SDSC), 46, 50
- Sanity checks, 204
- SAP. *See* Service Access Point
- SDSC. *See* San Diego Supercomputer Center
- Secondary servers, 201, 212, 218-19, 245
- Security, 261-77
  - administration, 273-77
  - Diffie-Hellman method, 266-67
  - finger protocol, 395
  - NFS, 264, 267-69, 275-77
  - NIS+, 235-39
  - performance, 276-77
  - RPC, 263-73, 276-77
- Selective acknowledgment (SACK), 86, 542
- Selectors, 319
- SEND command, 330
- Sending node, 77
- Sendmail, 341, 344
- Send state, 189
- Sequence checking, 211
- Sequence number, 78, 100-101, 542
- Serial Line Internet Protocol (SLIP), 32, 57
- Servers, 542
  - DNS, 240
  - interoperability, 448
  - lock manager, 308
  - NFS, 170-72
  - NIS, 218-20, 222-26, 231, 240-45, 254-56
  - NTP, 201, 298-212
  - recursion, 254-56
  - RPC, 183, 186, 189-93
  - security, 272
  - types, 243
- Service Access Point (SAP), 15-16
- Service classes, 387, 543
- Service procedure, 121
- Service protocols, 62
- Session, 271, 543
- Showmount command, 155-56
- Silly window syndrome, 82-83
- Simple Mail Transport Protocol (SMTP), 41, 217-18, 325-31, 545
  - changing roles, 328

- paths, 330-31
- steps, 326
- users, 328-30
- Simple Network Management Protocol (SNMP), 407-408, 412-15, 423-26, 545
- Single-binding control procedure, 190
- Site licenses, 296
- Slave servers, 222-26
- SLIP. *See* Serial Line Internet Protocol
- SMDS. *See* Switched Multi-megabit Data Service
- SMI. *See* Structure of management information
- SMTP. *See* Simple Mail Transport Protocol
- SNA. *See* System Network Architecture
- Snail mail, 344, 347, 545
- SNAP. *See* Subnetwork Access Point
- Sniffer Network Analyzer, 5, 16, 545
- SNMP. *See* Simple Network Management Protocol
- SOA. *See* Start of Authority
- Socket, 319, 545
- Soft mount, 153
- SONET. *See* Synchronous Optical Network
- Source quench message, 73, 87-88
- Split horizon, 93-94
- Staging, 176
- Start of Authority (SOA), 247
- Start state, 188
- Stateless protocol, 157, 166-68
- State machine, 188-89, 547
- Stock tool, 42, 146-47
- Storage processor, 174-76
- Stratum, 201
- Stream mode, 383
- STREAMS, 117-25, 548
  - components, 119-21
  - flow control, 121-23
  - input/output, 118-23
  - messages, 78, 120-23
  - multiplexer, 123
  - protocol, 117-18
- Structure files, 418, 548
- Structure of management information (SMI), 408
- Subdirectory field, 290
- Subnegotiation, 374-75
- Subnet, 67, 548
- Subnetwork access point (SNAP), 15-16
- Subnetworks, 9-57, 548
  - defined, 9, 54, 65-68
  - Internet, 42-53
  - mask, 68, 94
  - numbers, 94-95
  - radio, 53-57
  - SWAN, 36-42
  - token rings, 24-31
  - wide-area links, 31-36
  - see also* Ethernet
- Substitution, 290
- SunAccess, 442-43
- Sun-Barr host, 344, 347
- SunLink
  - BSC RJE, 440-41
  - Channel Gateway, 41, 88, 390, 436-38
  - SNA Peer-to-Peer, 441
  - 3270 products, 438-440, 444
- SunNet License Service, 296-97
- SunNet Manager, 108, 147, 408, 415-31
  - agents, 416-18
  - architecture, 408, 416, 420
  - database, 418-23
  - elements, 418-19
  - operation, 426-29
  - tools, 429-31
- Sun-3 system, 277
- Sun Unix. *See* Unix systems
- Sun Wide-Area Networks (SWAN), 36-42, 342-47, 442
- Support protocols, 69-84
- SVID. *See* System V Interface Definition
- SWAN. *See* Sun Wide-Area Networks

- Switched Multi-megabit Data Service, 35-36
- Symbolic links, 162
- Symmetric cryptography, 264
- Symmetric key systems, 262
- Symmetric modes, 205
- Symmetry, 372
- SYN. *See* Synchronize
- Synchronization. *See* Network Time Protocol
- Synchronize (SYN), 76, 78, 112, 549
- Synchronous bandwidth, 28-29
- Synchronous mode, 29-30, 549
- Synchronous Optical Network (SONET), 36, 50, 546
- Synchronous processing, 193
- Sync signal, 373
- Syntax, 409
- System V Interface Definition (SVID), 306-307
- System call, 78
- System Network Architecture (SNA), 435-52, 549
  - case studies, 442-52
  - DEC, 452-54
  - OSI, 452-55
  - overview, 435-36
  - PC, 455-57
  - SunLink products, 436-41
- Table, 223, 550
- TAI. *See* International Atomic Time
- Target token rotation time (TTRT), 28-29, 30, 553
- TCP. *See* Transmission Control Protocol
- Telnet, 371-77, 378, 551
  - NVT, 372-74
  - output discarding, 375-76
  - subnegotiation and options, 374-75
  - traffic, 377
  - workstation support, 376-77
- Telnet 3270, 438
- Terminal Node Controller (TNC), 53
- Terminals, 371-404
- Terminal server, 18
- Terminate and stay resident utilities, 317
- Text data set, 170-71
- TFTP. *See* Trivial File Transfer Protocol
- ThickWire, 18, 19, 551
- ThinWire, 18, 19, 551
- Threshold token rotation time (TRT), 29
- THT. *See* Token holding timer, 27
- Ticket, 271, 272
- Time-exceeded message, 73
- Timeout timer, 95
- TIME protocol, 200, 213, 400
- Time services. *See* Network Time Protocol
- Timestamps, 74, 201, 266, 272, 277
- Time synchronization, 200-204, 206-208
- Time-to-live (TTL), 63, 65, 73, 247, 255
- Tinygrams, 87
- TI-RPC, 117, 139, 193
- TLI. *See* Transport Level Interface
- TNC. *See* Terminal node controller
- Token holding timer (THT), 27, 551
- Token ring, 24-31, 552
- Token rotation timer (TRT), 29, 553
- T1 network, 36, 38, 550
- TOS. *See* Type of service
- Town and Country Building Society, 446-52
- TP Class 4 (TP4), 88
- TPO transport protocols, 88-89
- Trace fields, 334
- Trailers, 106-107
- Transaction IDs, 167, 396
- Transfer byte size, 382
- Transmission Control Protocol (TCP), 550, 552
  - external data processing, 143
  - interoperability, 435
  - limitations, 85-86, 140
  - networks, 62, 74-89, 106-13
  - NFS, 157
  - NTP, 200
  - PC-NFS, 320
  - resource location, 393-96

- REX, 310
- RPC, 134, 140
- security, 261
- services, 398-400
  - see also* File Transfer Protocol;  
Mail; Telnet
- Transmit procedure, 206
- Transport Level Interface (TLI), 117,  
123-25, 552
- Transport protocols, 62, 88-89, 117,  
137-39, 184-85, 552
- Transport Provider Interface, 123
- Traps, 192, 425, 552
- Tree, 409-10
- Triggered updates, 97
- Trivial File Transfer Protocol (TFTP),  
70, 273, 396-98
- Trojan horse list, 274-75
- TRT. *See* Threshold token rotation  
time
- TRT. *See* Token rotation timer
- TSRs. *See* Terminate and stay  
resident utilities
- TTL. *See* Time-to-live
- TTRT. *See* Target token rotation time
- Two-repeater rule, 19
- Type of Service (TOS), 69
  
- X.25 protocols, 34-35, 559
- X.400 environment, 325, 347-50,  
365-66, 560
- X.500 name service, 256, 347, 560
- X Windows System, 194, 377
  
- Yellow Pages, 217, 561
  - see also* Network Information  
Service
- Ypbind, 219
- Ypwhich command, 220
- Ypxfer utility, 42, 226
  
- Zimmerman, David, 396
- Zone data, 245







(continued from front flap)

*Analyzing Sun Networks* is an essential work for everyone interested in computer networks in general and Sun networks in particular. It will be especially helpful to MIS managers who need to understand the implications of TCP/IP, NFS, SNMP, and other key protocols.

#### **About the Author**

**CARL MALAMUD** is a consultant to many government agencies and major corporations, and conducts technical training seminars for many of the leading computer companies. He is the author of several books, including *INGRES: Tools for Building an Information Architecture* and the two companion volumes in this series, *Analyzing Novell Networks* and *Analyzing DECnet/OSI Phase V*.

## Books of Related Interest From Van Nostrand Reinhold

### **ANALYZING NOVELL NETWORKS**

By **Carl Malamud**, 360 pages, 7½ x 9¼, ISBN 0-442-00364-1

This book, the first in the Analyzing Networks series, provides a solid understanding of Novell's Netware and the underlying technology upon which Netware is built. This book contains a detailed analysis of the configuration of a Novell network, the underlying data links—ARCnet, Ethernet, and Token Ring—on which Novell is built, and the key network protocols, IPX and SPX. *Analyzing Novell Networks* also contains important information on key network services, including the Netware Core Protocols, the Message Handling Service, Remote Procedure Calls, STREAMS and the interconnection of Novell with other network architectures.

### **ANALYZING DECnet/OSI PHASE V**

By **Carl Malamud**, 544 pages, 7½ x 9¼, ISBN 0-442-00375-7

Here is the second in a series of books on Analyzing Networks which provides a technical introduction to this important new network architecture. With a clear focus on what DECnet/OSI Phase V does and how it relates to other parts in the network, it offers an architectural perspective of the system's design and its real-world implications. Readers will discover valuable information on different datalinks such as Ethernet and FDDI, services such as messaging and the Digital Naming Service, plus the Enterprise-Wide Management Architecture, an important DEC network management architecture for managing various networks from one operating system.

### **INGRES**

#### **Tools for Building an Information Architecture**

By **Carl Malamud**, 350 pages, 7½ x 9¼, ISBN 0-442-31800-6

"Practical and well-written introduction to Ingres and relational databases."—*Dr. Michael Stonebraker, Co-Founder, Ingres Corp.*

"A fascinating look at Ingres . . . a valuable source of information for anyone who wants to understand how a state-of-the-art database engine works."—*Paul E. Newton, President and CEO, Ingres Corp.*

